Systemy operacyjne 1

Zarządzanie pamięcią operacyjną

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 28 listopada 2024

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

1. Zagadnienia podstawowe
   1. Wiązanie adresów
   2. Ładowanie dynamiczne
   3. Łączenie dynamiczne
   4. Nakładki
2. Wymiana
3. Przydział ciągłych obszarów
   1. Przydział pojedynczego obszaru
   2. Przydział wielu obszarów
4. Stronicowanie
   1. Zasada działania
   2. Wspomaganie sprzętowe
   3. Strony współdzielone i ochrona
5. Segmentacja
   1. Zasada działania
   2. Wspomaganie sprzętowe
   3. Współdzielone segmenty i ochrona
6. Segmentacja stronicowana

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

1. Zagadnienia podstawowe
   1. Wiązanie adresów
   2. Ładowanie dynamiczne
   3. Łączenie dynamiczne
   4. Nakładki
2. Wymiana
3. Przydział ciągłych obszarów
   1. Przydział pojedynczego obszaru
   2. Przydział wielu obszarów
4. Stronicowanie
   1. Zasada działania
   2. Wspomaganie sprzętowe
   3. Strony współdzielone i ochrona
5. Segmentacja
   1. Zasada działania
   2. Wspomaganie sprzętowe
   3. Współdzielone segmenty i ochrona
6. Segmentacja stronicowana

Wstęp

Najprostszy model pamięci operacyjnej opisuje ją jako ciąg komórek (lokacji) jednakowego rozmiaru, z których każda posiada unikalny identyfikator. Rozmiar komórki najczęściej wynosi jeden bajt, w przypadku ogólnym jest natomiast wyrażony słowem binarnym. Identyfikator komórki nazywany *adresem* jest liczbą całkowitą, najczęściej liczbą naturalną, która w sposób jednoznaczny identyfikuje tę komórkę. Procesor jest połączony z pamięcią trzema magistralami: *magistralą sterowania*, *magistralą danych* i *magistralą adresową*. W komunikacji między tymi układami często również pośredniczy jednostka zarządzania pamięcią (ang. Memory Management Unit - MMU). Pamięć operacyjna stanowi podstawowy magazyn danych procesora. Każdy program, który podlega wykonaniu musi być w niej umieszczony. W dalszej części wykładu zajmiemy się sposobami umiejscowienia go w pamięci operacyjnej oraz tym co dzieje się z wytworzonymi przez niego adresami na drodze procesor - pamięć. Nie będzie nas natomiast interesował sposób wytwarzania tych adresów.

Wiązanie adresów

Program komputerowy przed wykonaniem znajduje się najczęściej na dysku twardym lub innym nośniku w postaci wykonywalnego pliku binarnego. Na zlecenie użytkownika lub określonego procesu może on zostać załadowany do pamięci operacyjnej i wykonany. W systemach wsadowych najpierw trafia do *kolejki wejściowej* na dysku. Jeśli program będzie wykonywany wielokrotnie, to może się okazać, że za każdym razem będzie ładowany do innego miejsca w pamięci. Pamięć komputera zaczyna się zazwyczaj od adresu zero, natomiast pamięć programu, czyli ten obszar pamięci fizycznej, który został przydzielony mu przez system operacyjny, najczęściej nie. Konieczne jest więc przeprowadzenie *wiązania* adresów wytwarzanych przez program z adresami fizycznymi. Ta operacja może zostać przeprowadzona na różnych etapach przygotowania programu do wykonania:

Wiązanie adresów

- **Etap kompilacji** W kodzie źródłowym adresy są wyrażane w *postaci symbolicznej*, czyli są to *nazwy zmiennych, podprogramów, itp.*. Jeśli kompilator dysponuje informacjami, gdzie wygenerowany przez niego kod wynikowy będzie umieszczony w pamięci komputera (tak się dzieje jedynie w przypadku systemów, w których może być uruchamiany jedynie jeden proces użytkownika), to zamienia adresy symboliczne bezpośrednio na adresy fizyczne, nazywane *adresami bezwzględnymi*.
- **Etap ładowania** Jeśli informacje na temat umiejscowienia programu w pamięci nie są znane na etapie kompilacji, to kompilator przekształca adresy symboliczne na *adresy względne*, inaczej *relokowalne* liczone względem początku generowanego przez niego bloku kodu. W ten sposób powstaje kod *przemieszczalny* lub *relokowalny*. Wiązania adresów względnych z adresami fizycznymi dokonuje program ładujący.
- **Etap wykonania**. Jeśli procesor jest wyposażony w odpowiedni sprzęt, to możliwe jest przemieszczanie programu w pamięci podczas jego wykonania, np.: w celu zrobienia miejsca w pamięci innemu programowi, który będzie do niej załadowany. Tę metodę określa się również mianem *dynamicznego wiązania adresów*.

Wiązanie adresów

- **Etap kompilacji** W kodzie źródłowym adresy są wyrażane w *postaci symbolicznej*, czyli są to *nazwy zmiennych, podprogramów, itp.*. Jeśli kompilator dysponuje informacjami, gdzie wygenerowany przez niego kod wynikowy będzie umieszczony w pamięci komputera (tak się dzieje jedynie w przypadku systemów, w których może być uruchamiany jedynie jeden proces użytkownika), to zamienia adresy symboliczne bezpośrednio na adresy fizyczne, nazywane *adresami bezwzględnymi*.
- **Etap ładowania** Jeśli informacje na temat umiejscowienia programu w pamięci nie są znane na etapie kompilacji, to kompilator przekształca adresy symboliczne na *adresy względne*, inaczej *relokowalne* liczone względem początku generowanego przez niego bloku kodu. W ten sposób powstaje kod *przemieszczalny* lub *relokowalny*. Wiązania adresów względnych z adresami fizycznymi dokonuje program ładujący.
- **Etap wykonania**. Jeśli procesor jest wyposażony w odpowiedni sprzęt, to możliwe jest przemieszczanie programu w pamięci podczas jego wykonania, np.: w celu zrobienia miejsca w pamięci innemu programowi, który będzie do niej załadowany. Tę metodę określa się również mianem *dynamicznego wiązania adresów*.

Wiązanie adresów

- **Etap kompilacji** W kodzie źródłowym adresy są wyrażane w *postaci symbolicznej*, czyli są to *nazwy zmiennych, podprogramów, itp.*. Jeśli kompilator dysponuje informacjami, gdzie wygenerowany przez niego kod wynikowy będzie umieszczony w pamięci komputera (tak się dzieje jedynie w przypadku systemów, w których może być uruchamiany jedynie jeden proces użytkownika), to zamienia adresy symboliczne bezpośrednio na adresy fizyczne, nazywane *adresami bezwzględnymi*.
- **Etap ładowania** Jeśli informacje na temat umiejscowienia programu w pamięci nie są znane na etapie kompilacji, to kompilator przekształca adresy symboliczne na *adresy względne*, inaczej *relokowalne* liczone względem początku generowanego przez niego bloku kodu. W ten sposób powstaje kod *przemieszczalny* lub *relokowalny*. Wiązania adresów względnych z adresami fizycznymi dokonuje program ładujący.
- **Etap wykonania**. Jeśli procesor jest wyposażony w odpowiedni sprzęt, to możliwe jest przemieszczanie programu w pamięci podczas jego wykonania, np.: w celu zrobienia miejsca w pamięci innemu programowi, który będzie do niej załadowany. Tę metodę określa się również mianem *dynamicznego wiązania adresów*.

Wiązanie adresów

- **Etap kompilacji** W kodzie źródłowym adresy są wyrażane w *postaci symbolicznej*, czyli są to *nazwy zmiennych, podprogramów, itp.*. Jeśli kompilator dysponuje informacjami, gdzie wygenerowany przez niego kod wynikowy będzie umieszczony w pamięci komputera (tak się dzieje jedynie w przypadku systemów, w których może być uruchamiany jedynie jeden proces użytkownika), to zamienia adresy symboliczne bezpośrednio na adresy fizyczne, nazywane *adresami bezwzględnymi*.
- **Etap ładowania** Jeśli informacje na temat umiejscowienia programu w pamięci nie są znane na etapie kompilacji, to kompilator przekształca adresy symboliczne na *adresy względne*, inaczej *relokowalne* liczone względem początku generowanego przez niego bloku kodu. W ten sposób powstaje kod *przemieszczalny* lub *relokowalny*. Wiązania adresów względnych z adresami fizycznymi dokonuje program ładujący.
- **Etap wykonania**. Jeśli procesor jest wyposażony w odpowiedni sprzęt, to możliwe jest przemieszczanie programu w pamięci podczas jego wykonania, np.: w celu zrobienia miejsca w pamięci innemu programowi, który będzie do niej załadowany. Tę metodę określa się również mianem *dynamicznego wiązania adresów*.

Ładowanie dynamiczne

Jeśli program jest wygenerowany w postaci przemieszczalnej, to można zastosować technikę pozwalającą na oszczędzanie miejsca w pamięci operacyjnej. Można ów program podzielić na podprogramy, z których każdy będzie rezydował w osobnym pliku binarnym na dysku. Do pamięci jest ładowany jedynie program główny. Jeżeli zajdzie konieczność wywołania któregoś z podprogramów, to najpierw sprawdzane jest, czy znajduje się on w pamięci operacyjnej, jeśli nie to wywoływany jest program ładujący, który go tam umieści. Dzięki tej technice podprogramy są ładowane do pamięci tylko wtedy, gdy są potrzebne.

Łączenie dynamiczne

Łączenie jest jednym z etapów translacji programu z kodu źródłowego na kod wynikowy. Polega ono na włączeniu wszystkich niezbędnych bibliotek do kodu wynikowego kompilowanego programu. W systemie są zazwyczaj biblioteki, z których korzysta wiele procesów. W wyniku zwykłego łączenia każdy z tych procesów otrzymuje swoją kopię kodu tych bibliotek, którą należy oczywiście umieścić w pamięci operacyjnej. *Łączenie dynamiczne* pozwala uniknąć narzutów pamięci, które powstają w wyniku zwykłego łączenia. W kodach wynikowych programów stosujących tę technikę, w miejscu odwołania do podprogramu znajdującego się w bibliotece zewnętrznej znajduje się tzw. *zakładka*, czyli fragment kodu, który pozwala znaleźć w pamięci operacyjnej podprogram z odpowiedniej *biblioteki współdzielonej*. Jeśli ta biblioteka nie znajduje się w pamięci to jest ładowana. Usuwanie biblioteki przeprowadzane jest tylko wtedy, gdy jest to konieczne i żaden proces nie korzysta z niej. Biblioteki współdzielone są wyposażane w numer wersji, dzięki temu każdy proces korzystający z łączenia dynamicznego może określić, czy w systemie jest potrzebna mu wersja biblioteki. Pliki w których znajduje się kod binarny bibliotek są plikami wykonywalnymi, mogą być one zgromadzone w jednym miejscu na dysku (w jednym katalogu) bądź mogą być umieszczone w katalogu z kodem binarnym programu. W większości współczesnych systemów zakładka odwołuje się do procesu systemowego, który jest odpowiedzialny za zarządzanie bibliotekami współdzielonymi i ich udostępnianie procesom. W systemie Windows biblioteki współdzielone określane są skrótem DLL (ang. Dynamic Linked Libraries), natomiast w systemach uniksowych przez skrót SO (ang. Shared Object).

Nakładki

Technika *nakładania* pozwala wykonać program, który ze względu na swe rozmiary nie może być w całości umieszczony w pamięci operacyjnej. Przypomina ona w działaniu technikę dynamicznego ładowania. *Nakładka* jest fragmentem obrazu programu komputerowego, czyli takiej postaci programu jaka jest umieszczana w pamięci komputera. W czasie działania programu jest ładowana do pamięci ta jego część która jest zawsze niezbędna i która zawiera kod zarządzający nakładkami, oraz potrzebna w danej chwili nakładka. Jeśli ta nakładka przestanie być potrzebna to kod wykonujący nakładanie wysyła ją na dysk i ładuje do pamięci tę, która jest bieżąco potrzebna. Nakładanie nie wymaga wsparcia ze strony systemu operacyjnego i może w całości być realizowane na poziomie programu użytkownika. Oznacza to, że programista tworzący taki program musi samodzielnie zaimplementować tę technikę, co wymaga stosunkowo dobrej wiedzy na temat rozmieszczenia programu w pamięci operacyjnej.

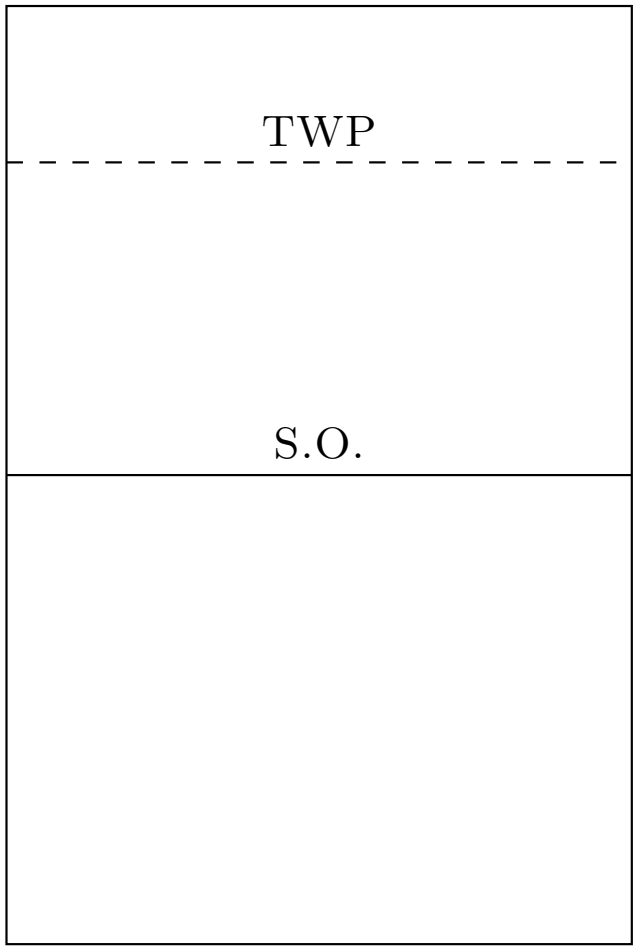
Prosta wymiana

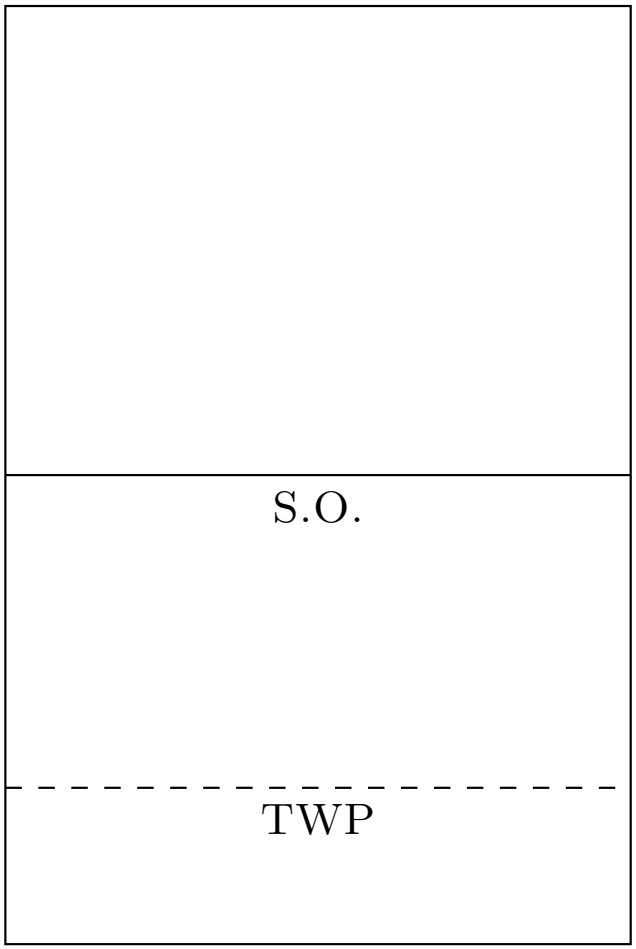
Jeśli istnieje konieczność zrealizowania przetwarzania wielozadaniowego w systemie komputerowym o ograniczonej pojemności pamięci, to można posłużyć się techniką prostej wymiany. Dzięki niej jedynie proces, który jest w stanie aktywnym musi znajdować się w pamięci operacyjnej. Pozostałe procesy mogą być umieszczone w szybkiej *pamięci pomocniczej* umiejscowionej na dysku twardym. Przełączanie kontekstu obejmuje przeniesienie procesu, który utracił procesor do pamięci pomocniczej i załadowanie do pamięci operacyjnej procesu, który został wybrany przez planistę do wykonania. W systemach z priorytetowym szeregowaniem proces o wyższym priorytecie może wywłaszczać z pamięci operacyjnej proces o niższym priorytecie, przy czym ten ostatni wraca do niej po wykonaniu ważniejszego zadania. Tę odmianę wymiany nazywamy *zwijaniem i rozwijaniem*. Wadą tego rozwiązania jest szybkość działania. Pamięć pomocnicza jest wielokrotnie wolniejsza od pamięci operacyjnej, dlatego przy wymianie potrzebne są informacje nie o tym ile pamięci zostało procesowi przydzielone, ale ile z tej pamięci faktycznie wykorzystuje. Pozwala to uniknąć transferu niepotrzebnych danych. Jeśli system nie stosuje dynamicznego wiązania adresów to przywracany proces musi trafić dokładnie w to samo miejsce w pamięci, które zajmował przed wymianą. Wymianie nie mogą podlegać procesy, które czekają na realizację operacji wejścia-wyjścia dla których bufory przydzielono w obszarze pamięci tych procesów. Prosta wymiana była stosowania w pierwszych systemach uniksowych w wersji BSD. Zastosowana w nich realizacja tej techniki była jednak doskonalsza od oryginalnej idei bo procesy były wymieniane tylko wtedy, gdy brakowało miejsca w pamięci operacyjnej.

Przydział pojedynczego obszaru

W najprostszym scenariuszu przydziału pamięci cała dostępna pamięć fizyczna jest przydzielana pojedynczemu procesowi, który oprócz określonych zdań wykonuje wszystkie prace charakterystyczne dla systemu operacyjnego, jak np. obsługa przerwań. Takie rozwiązanie jest spotykane najczęściej w systemach bazujących na mikrokontrolerach. Trochę bardziej skomplikowany jest przypadek, w którym w pamięci rezyduje system operacyjny i dokładnie jeden proces użytkownika. Ponieważ system przerwań stanowi część systemu operacyjnego i musi być wraz z nim chroniony, to pamięć dla systemu operacyjnego jest przydzielana tam, gdzie projektanci procesora określili położenie tablicy wektorów przerwań. Często jest to dolna część pamięci, rozpoczynająca się od adresu 0, rzadziej pamięć górna, kończąca się maksymalnym adresem. Decyzja o położeniu TWP może być też pozostawiana przez twórców sprzętu programiście systemowemu. Problem przydziału pamięci procesowi użytkownika jest trudniejszy. Kod tego procesu może być umieszczony bezpośrednio za kodem systemu operacyjnego. Takie rozwiązanie powoduje problemy, kiedy system operacyjny chce przydzielić sobie więcej pamięci, np. na bufory wejścia-wyjścia lub gdy korzysta z *kodu przejściowego*, tj. takiego kodu, który nie rezyduje na stałe w pamięci, a jest ładowany w razie potrzeby. Doraźne rozwiązanie polega na umieszczeniu systemu operacyjnego na początku pamięci (w pamięci dolnej), a procesu użytkownika w pamięci górnej, tak aby ostatni adres programu użytkownika był jednocześnie ostatnim adresem w pamięci operacyjnej. Dzięki temu pośrodku pamięci operacyjnej powstaje obszar pamięci wolnej, z którego mogą dowolnie korzystać oba procesy. Bardziej ogólne rozwiązanie zostanie opisane na następnej planszy.

Przydział pojedynczego obszaru

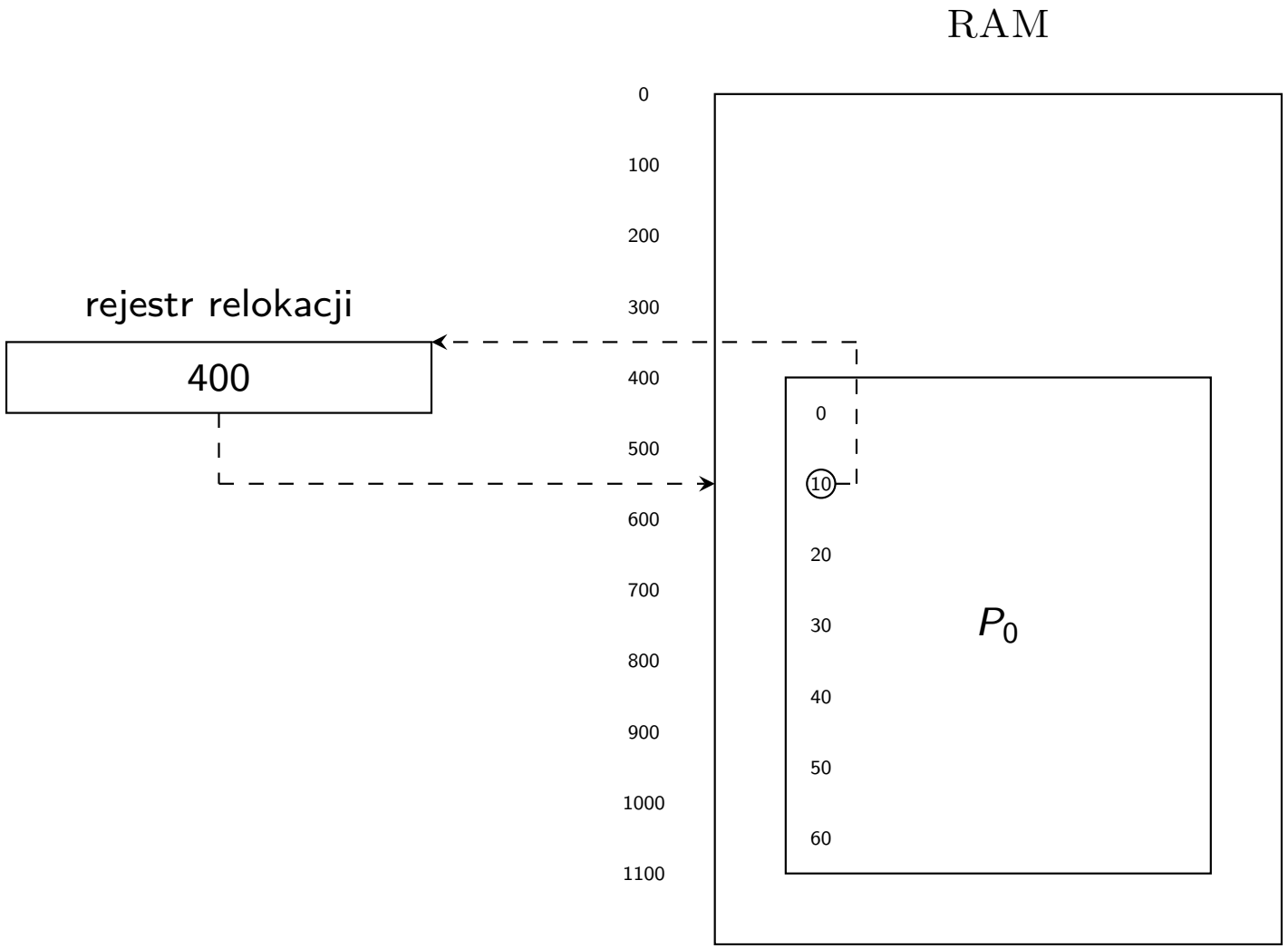




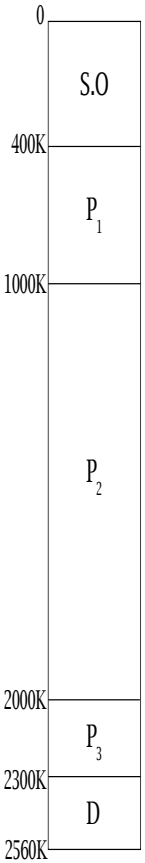
Pamięć logiczna i fizyczna

Zauważmy, że w opisanym wyżej schemacie, do zapewnienia ochrony pamięci wystarczy zastosować tylko rejestr bazowy, opisany na wcześniejszych wykładach. Rejestr graniczny jest zbędny, bo jednemu procesowi można oddać do dyspozycji całą pamięć znajdującą się za adresem bazowym. Jeśli system operacyjny podczas pracy nie przydziela sobie więcej pamięci operacyjnej, to wartość rejestru bazowego jest statyczna. Ponadto można w programie użytkownika umieścić adresy bezwzględne już na etapie kompilacji. Jeżeli jednak takie przydziały następują, to trzeba zastosować adresy względne i dynamicznie zmieniać wartość rejestru bazowego. Przy takim podejściu zastosowanie wiązania adresów tylko podczas ładowania jest niewystarczające. Należałoby przerywać pracę programu i ponownie go załadować, uwzględniając nową wartość rejestru bazowego za każdym razem kiedy system operacyjny przydzieli sobie pamięć. Aby uniknąć takiego problemu stosuje się wiązanie dynamiczne. Rejestr bazowy jest wykorzystywany jako *rejestr relokacji* lub *rejestr przemieszczenia*, którego wartość jest dodawana do każdego adresu wygenerowanego przez proces użytkownika. Należy zauważyć, że mamy teraz do czynienia z dwoma obrazami pamięci. Pierwszy to pamięć rzeczywista, nazywana *pamięcią fizyczną* i związana z nią *fizyczna przestrzeń adresowa*. Jest to cała pamięć, którą dysponuje system komputerowy i którą zarządza system operacyjny. Drugim obrazem pamięci jest pamięć taka, jaką widzi ją proces użytkownika. Tę pamięć nazywamy *pamięcią logiczną* i jest z nią związana *logiczna przestrzeń adresowa*. Proces używa *adresów względnych* począwszy od adresu o wartości zero. Zanim te adresy dotrą do pamięci operacyjnej, to dodawana jest do nich zawartość rejestru relokacji, dzięki czemu adresowane są prawidłowe komórki. Innymi słowy proces posługuje się adresami względnymi z zakresu [0,max], gdzie max oznacza adres względny o największej dla tego procesu wartości, a te adresy są tłumaczone dynamicznie na adresy fizyczne z przedziału [R,R+max], gdzie R to bieżąca wartość rejestru relokacji. Jeśli chcemy przemieścić proces w pamięci, to wystarczy go skopiować do obszaru docelowego i zmienić zawartość rejestru bazowego.

Pamięć logiczna i fizyczna

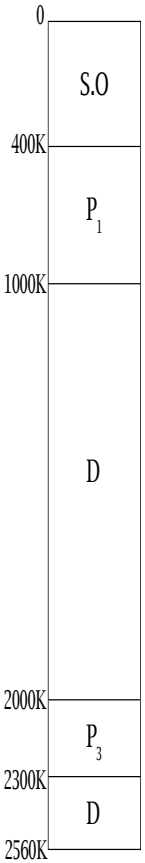


Przydział wielu obszarów



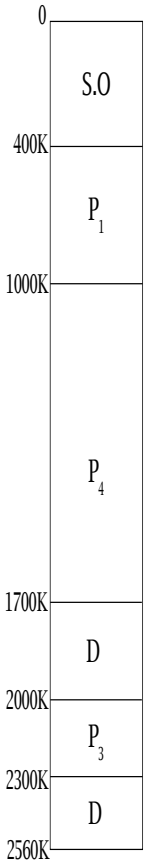
Wyobraźmy sobie, że w kolejce wejściowej systemu, który stosuje szeregowanie długoterminowe algorytmem FCFS, jest pięć procesów, których zapotrzebowania na pamięć wynoszą odpowiednio: 600KB, 1000KB, 300KB, 700KB i 500KB. Dostępnych jest tylko 2560KB pamięci operacyjnej. Można ją przydzielić od razu pierwszemu, drugiemu i trzeciemu procesowi. W pamięci zostanie miejsce wolne, nazywane krótko *dziurą*, które będzie niewystarczające, dla żadnego z pozostałych procesów. Będą musiały one poczekać, aż zakończy się jeden z trzech procesów, które są już w pamięci.

Przydział wielu obszarów



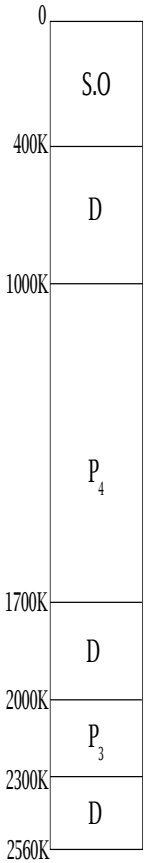
Załóżmy, że swoją pracę jako pierwszy skończył proces drugi. W pamięci powstają zatem dwa ciągłe obszary wolne. Okazuje się, że dziurę po procesie P_2 można przydzielić tylko jednemu z procesów, które czekają na przydział pamięci. Tym procesem będzie proces P_4.

Przydział wielu obszarów



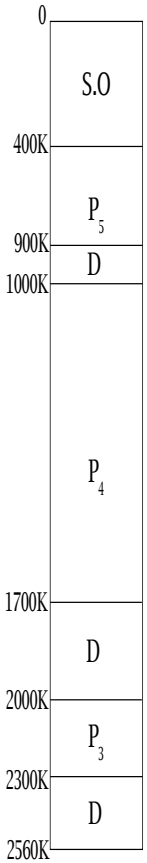
Zauważmy, że po tym przydziale w pamięci operacyjnej nadal istnieją dwie dziury, o sumarycznej wielkości 560KB. Ta wielkość spełnia wymagania programu piątego (500KB), ale ponieważ te dziury nie tworzą obszaru spójnego, to nie można ich przydzielić temu procesowi.

Przydział wielu obszarów



Przyjmijmy, że następnym procesem, który zakończy swą pracę będzie proces P_1. Po jego zakończeniu w pamięci operacyjnej pojawia się ciągły obszar wolny o wielkości 600KB, co pozwala na spełnienie zapotrzebowania na pamięć ze strony procesu piątego. Z tej dziury procesowi P_5 zostanie przydzielone dokładnie 500KB.

Przydział wielu obszarów



Zauważmy, że po wykonaniu ostatniego z przydziałów w pamięci operacyjnej powstają trzy niespójne wolne obszary o łącznej pojemności 660KB.

Strategie przydziału

W opisywanym schemacie mamy do czynienia z *dynamicznym przydziałem* pamięci. System operacyjny utrzymuje ewidencję wolnych oraz zajętych obszarów pamięci i na bieżąco podejmuje decyzję, który z wolnych obszarów przydzielić czekającym procesom. To planowanie wymaga określenia strategii wyboru dziury, jeśli jest ich wiele. Oto trzy najpopularniejsze strategie:

- **Pierwsza pasująca**-przydzielana jest pierwsza dziura, która spełnia wymagania procesu co do rozmiaru. Jej poszukiwanie może zawsze się rozpoczynać od początku wykazu miejsc wolnych lub od miejsca ostatniego przydziału.
- **Najlepiej pasująca**-przydziela się dziurę, która dokładnie spełnia wymagania co do rozmiaru, lub dla której różnica między jej rozmiarem, a rozmiarem wymaganym jest najmniejsza spośród pozostałych dziur.
- **Najgorzej pasująca**-przydziela się największą dziurę w systemie. Zakłada się, że proces zażąda w czasie wykonania dodatkowej pamięci. Przy zastosowaniu tej strategii istnieje duże prawdopodobieństwo, że żądanie to zostanie od razu zrealizowane.

Symulacje wykazały, że najlepsze rezultaty uzyskuje się przy zastosowaniu strategii **Pierwsza pasująca** i **Najlepiej pasująca**. Z praktycznego punktu widzenia ta pierwsza jest lepsza ponieważ nie tworzy dużych narzutów czasowych. Więcej informacji na temat strategii przydziału można znaleźć w książce D.E.Knuth'a „Sztuka programowania" tom I.

Fragmentacja

Zjawisko fragmentacji pamięci występuje powszechnie w systemach, które stosują dynamiczny przydział pamięci. Rozróżniamy dwa rodzaje tego zjawiska:

- **Fragmentacja zewnętrzna**-występuje wtedy, gdy w pamięci operacyjnej istnieje wiele obszarów wolnych, których sumaryczna wielkość pozwalałaby na spełnienie żądania przydziału pamięci przez proces, ale każdy z osobna z tych obszarów ma zbyt małą pojemność, żeby to było możliwe.
- **Fragmentacja wewnętrzna**-występuje wtedy, gdy procesowi przydzielane jest więcej pamięci, niż on potrzebuje. Ta dodatkowa pamięć nie będzie nigdy przez niego wykorzystana. Przydział taki może być podyktowany względami ekonomicznymi: nie jest opłacalne utrzymywanie 2 bajtowej dziury w pamięci, jeśli należy zapamiętać o wiele więcej informacji o niej.

Fragmentacja podlega regule 50%, tzn. po dużej liczbie cykli przydziałów i zwolnień będziemy w stanie przydzielić tylko połowę wolnej pamięci. Możemy zapobiegać temu zjawisku stosując *upakowanie* pamięci, tzn. okresowe przemieszczanie procesów, tak aby w pamięci powstał jeden ciągły obszar. System operacyjny powinien tak wykonywać tę operację, aby czas potrzebny na jej wykonanie był jak najmniejszy. Ta technika może być stosowana w systemach, w których możliwa jest relokacja. Jeśli oprócz procesów przemieszczalnych w pamięci są procesy z adresami bezwzględnymi to można tę technikę połączyć z prostą wymianą.

Fragmentacja

Zjawisko fragmentacji pamięci występuje powszechnie w systemach, które stosują dynamiczny przydział pamięci. Rozróżniamy dwa rodzaje tego zjawiska:

- **Fragmentacja zewnętrzna**-występuje wtedy, gdy w pamięci operacyjnej istnieje wiele obszarów wolnych, których sumaryczna wielkość pozwalałaby na spełnienie żądania przydziału pamięci przez proces, ale każdy z osobna z tych obszarów ma zbyt małą pojemność, żeby to było możliwe.
- **Fragmentacja wewnętrzna**-występuje wtedy, gdy procesowi przydzielane jest więcej pamięci, niż on potrzebuje. Ta dodatkowa pamięć nie będzie nigdy przez niego wykorzystana. Przydział taki może być podyktowany względami ekonomicznymi: nie jest opłacalne utrzymywanie 2 bajtowej dziury w pamięci, jeśli należy zapamiętać o wiele więcej informacji o niej.

Fragmentacja podlega regule 50%, tzn. po dużej liczbie cykli przydziałów i zwolnień będziemy w stanie przydzielić tylko połowę wolnej pamięci. Możemy zapobiegać temu zjawisku stosując *upakowanie* pamięci, tzn. okresowe przemieszczanie procesów, tak aby w pamięci powstał jeden ciągły obszar. System operacyjny powinien tak wykonywać tę operację, aby czas potrzebny na jej wykonanie był jak najmniejszy. Ta technika może być stosowana w systemach, w których możliwa jest relokacja. Jeśli oprócz procesów przemieszczalnych w pamięci są procesy z adresami bezwzględnymi to można tę technikę połączyć z prostą wymianą.

Fragmentacja

Zjawisko fragmentacji pamięci występuje powszechnie w systemach, które stosują dynamiczny przydział pamięci. Rozróżniamy dwa rodzaje tego zjawiska:

- **Fragmentacja zewnętrzna**-występuje wtedy, gdy w pamięci operacyjnej istnieje wiele obszarów wolnych, których sumaryczna wielkość pozwalałaby na spełnienie żądania przydziału pamięci przez proces, ale każdy z osobna z tych obszarów ma zbyt małą pojemność, żeby to było możliwe.
- **Fragmentacja wewnętrzna**-występuje wtedy, gdy procesowi przydzielane jest więcej pamięci, niż on potrzebuje. Ta dodatkowa pamięć nie będzie nigdy przez niego wykorzystana. Przydział taki może być podyktowany względami ekonomicznymi: nie jest opłacalne utrzymywanie 2 bajtowej dziury w pamięci, jeśli należy zapamiętać o wiele więcej informacji o niej.

Fragmentacja podlega regule 50%, tzn. po dużej liczbie cykli przydziałów i zwolnień będziemy w stanie przydzielić tylko połowę wolnej pamięci. Możemy zapobiegać temu zjawisku stosując *upakowanie* pamięci, tzn. okresowe przemieszczanie procesów, tak aby w pamięci powstał jeden ciągły obszar. System operacyjny powinien tak wykonywać tę operację, aby czas potrzebny na jej wykonanie był jak najmniejszy. Ta technika może być stosowana w systemach, w których możliwa jest relokacja. Jeśli oprócz procesów przemieszczalnych w pamięci są procesy z adresami bezwzględnymi to można tę technikę połączyć z prostą wymianą.

Ochrona

Jeśli dopuszczamy relokację procesów podczas wykonania, to poznany na drugim wykładzie schemat prostej ochrony pamięci należy trochę zmodyfikować. Otóż najpierw należy sprawdzić, czy adres logiczny wygenerowany przez program nie jest większy niż zawartość rejestru granicznego, a następnie, jeśli ten test się powiódł dodać do niego zawartość rejestru przemieszczenia. Celem zmniejszenia fragmentacji programy dzieli się na części. Najczęściej są to dwie części: zawierająca kod i zawierająca dane. Do ochrony tych części służą osobne pary rejestrów bazowych i granicznych. Jeśli chcemy wykonać podział programów na więcej części, to musimy mieć odpowiednio więcej par wymienionych rejestrów. Zastosowanie zwielokrotnionych rejestrów bazowych i granicznych pozwala również na określenie sposobu dostępu do danego fragmentu pamięci (np. tylko odczyt dla stałych) lub współdzielenie tych fragmentów przez kilka procesów (np. obszar kodu).

Stronicowanie

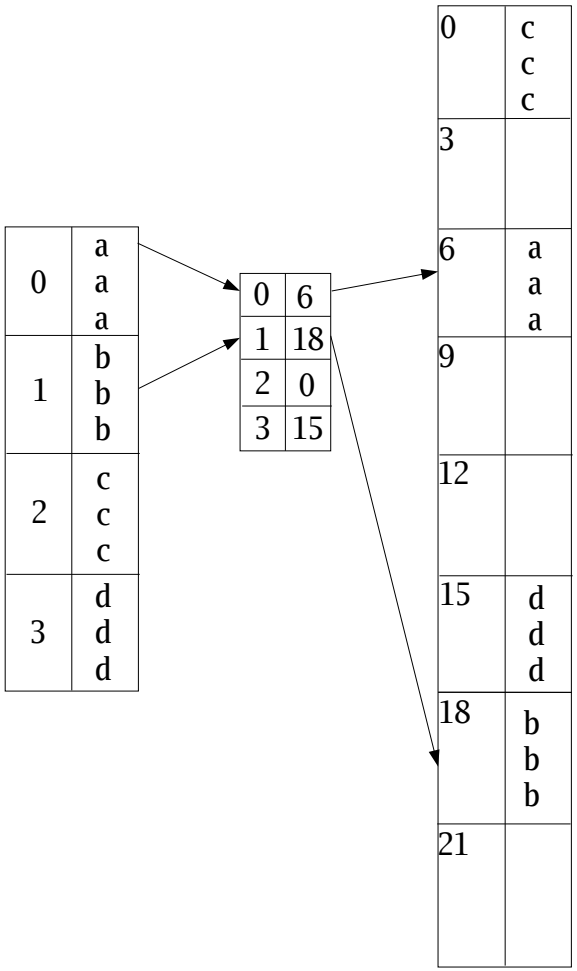
Stronicowanie (ang. paging) jest systemem zarządzania pamięcią, który rozwiązuje problem zewnętrznej fragmentacji pamięci pozwalając, aby pamięć przydzielana była *nieciągła*. Jest ono również w wielu wypadkach punktem wyjścia do implementacji bardziej zaawansowanych i nowoczesnych schematów zarządzania pamięcią. W każdym przypadku stronicowanie wymaga wspomagania sprzętowego. Stronicowanie jest jedną z możliwych realizacji przydziału dynamicznego pamięci. Dodatkowo można je łączyć z prostą wymianą[1].

[1] Bardziej efektywnym rozwiązaniem jest stronicowanie na żądanie, które będzie omawiane na następnych wykładach

Zasada działania

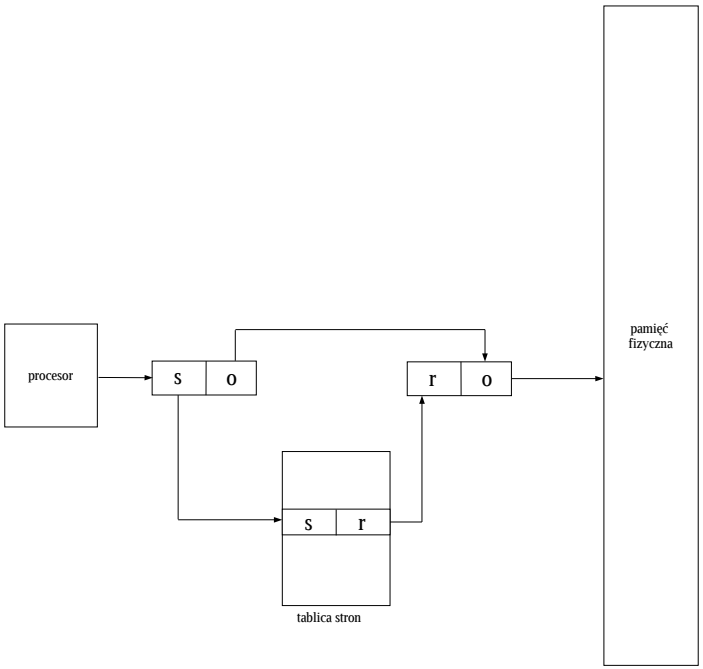
W stronicowaniu pamięć logiczna procesu jest podzielona na fragmenty o takiej samej wielkości, wyrażonej potęgą dwójki (zwykle od 512 do 2048 bajtów), nazywane *stronami*. Pamięć fizyczna jest podzielona na *ramki* nazywane również *stronami fizycznymi*, które mają ten sam rozmiar co strony. Również pamięć pomocnicza podzielona jest na obszary wielkości ramek nazywane *blokami*. Adres logiczny składa się z dwóch części: *numeru strony* oraz *przemieszczenia* względem początku strony. Podczas ładowania z pamięci pomocniczej do operacyjnej każda strona zostaje umieszczona w osobnej ramce. Rozmieszczenie to jest odwzorowywane w *tablicy stron*. Numery stron są traktowane jako indeksy tej tablicy, natomiast elementy zawierają adresy bazowe ramek, w których te strony zostały umieszczone. Translacja adresu logicznego na fizyczny wymaga odnalezienia w tablicy stron elementu o określonym przez numer strony indeksie, odczytaniu z niego adresu bazowego ramki i zastąpieniu nim numeru strony w adresie.

Zasada działania



Aby proces mógł być załadowany do pamięci musi jedynie istnieć odpowiednia liczba wolnych ramek dla niego. Strony w ramkach nie muszą być umieszczane po kolei, fizycznie pamięć procesu nie musi być ciągła. Taka sytuacja jest przedstawiona na ilustracji obok. Po lewej stronie znajduje się pamięć logiczna, w środku tablica stron, a po prawej pamięć fizyczna. Stronicowanie eliminuje całkowicie fragmentację zewnętrzną, ale nie jest odporne na fragmentację wewnętrzną. Ta fragmentacja dotyczy ostatniej ramki przydzielonej procesowi i w skrajnych przypadkach może wynosić rozmiar strony minus jeden bajt.

Translacja adresów



Aby stosować stronicowanie system musi być wyposażony w jednostkę MMU, która umożliwia translację adresów według przedstawionego obok schematu. Litera *s* oznacza numer strony, litera *r* adres bazowy ramki. Konieczność translacji adresu logicznego powoduje, że dostęp do pamięci odbywa się wolniej niż w systemach gdzie nie ma dynamicznego wiązania adresów.

Tablica stron

Każdy proces ma swoją tablicę stron, która jest umieszczona w obszarze pamięci systemu operacyjnego. Procesor jest wyposażony w *rejestr bazowy tablicy stron*[2], zawierający adres początku tablicy stron aktywnego procesu. Zawartość tego rejestru ulega oczywiście wymianie podczas przełączania procesów. Ponieważ każde odwołanie się przez proces użytkownika do pamięci operacyjnej pociąga za sobą konieczność translacji adresów, to programistom systemowym zależy, aby czas dostępu do tablicy stron był jak najkrótszy. Z pomocą przychodzą im projektanci sprzętu, umieszczając w procesorach *rejestry TLB* (ang. Translation Lookaside Buffers) nazywane w literaturze polskiej *rejestrami asocjacyjnymi*, a nawet *asocjacyjnymi buforami antycypacji translacji*. Jeśli tablica stron jest niewielkich rozmiarów, to można ją całkowicie w tych rejestrach umieścić, a ponieważ są one zbudowane z szybkich układów, to znalezienie adresu bazowego ramki na podstawie numeru strony jest wykonywane wielokrotnie szybciej niż przy bezpośrednim dostępie do pamięci. Jeśli tablica stron jest większa niż pojemność buforów, to przechowuje się w nich tylko te elementy tablicy stron, które są potrzebne. Jeśli jakiś element przestaje być potrzebny, to zostaje wymieniony na inny. Ta wymiana przebiega według następującego schematu: najpierw sprawdzane jest, czy rejestr asocjacyjny zawiera żądany numer strony, jeśli tak, to pobierany jest na jego podstawie adres bazowy ramki i następuje sięgnięcie do określonego fragmentu pamięci. Jeśli nie, to należy znaleźć w pamięci tablicę stron, odczytać z niej adres ramki (uaktualniając przy okazji rejestry TLB) i dopiero wtedy można sięgnąć do pamięci. Efektywny czas dostępu do pamięci zależy więc od *współczynnika trafień* (ang. hit ratio) i można go policzyć według wzoru:
$t_{ema} = h \cdot (t_{TLBa} + t_{ma}) + (1 - h) \cdot (t_{TLBa} + 2 \cdot t_{ma})$,
gdzie h jest współczynnikiem trafień, który jest wartością wyskalowaną od 0 do 1, t_{ema} jest efektywnym czasem dostępu do pamięci, t_{TLBa} czasem dostępu do rejestrów TLB, a t_{ma} czasem dostępu do pamięci. Współczesne platformy sprzętowe umożliwiają zarządzanie TLB z poziomu systemu operacyjnego.

[2] Uwaga: W opisie zakładamy, że MMU jest częścią procesora.

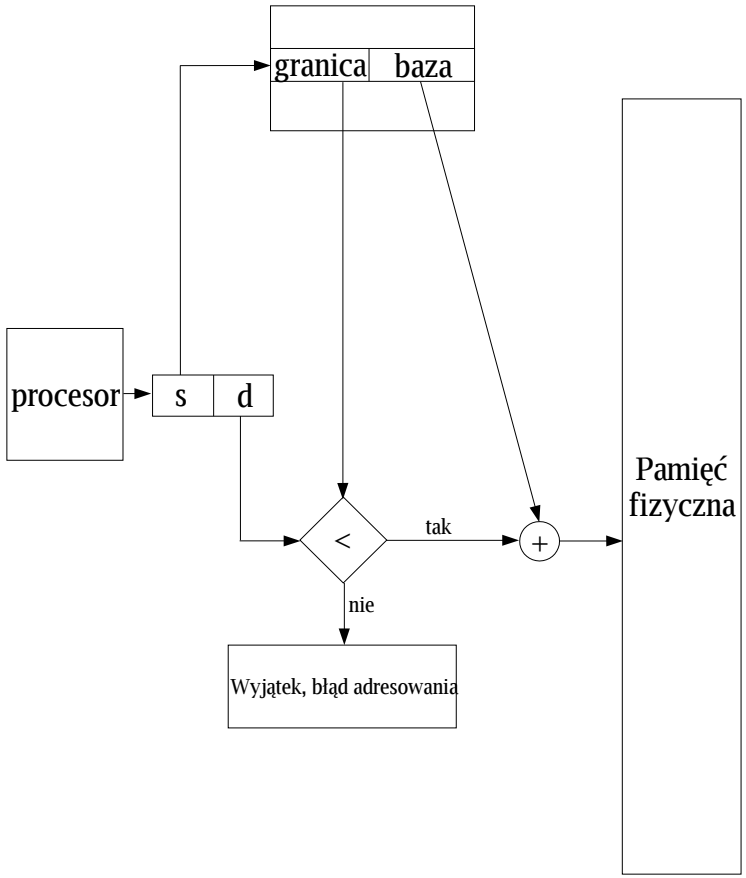
Strony współdzielone i ochrona

W systemach stosujących stronicowanie można dopuścić możliwość współdzielenia stron przez wiele procesów. Jeśli mamy uruchomionych wiele instancji tego samego programu, to korzystnie jest im pozwolić współdzielić strony z kodem, o ile ten kod jest *kodem wznawialnym*, nazywanym również *współużywalnym* (ang. reentrant). Aby kod był wznawialny wymaga się aby nie podlegał modyfikacją podczas wykonania. Warunek ten jest spełniany w większości współczesnych systemów. Łatwo jest również zapewnić ochronę stron. Jeśli dany numer strony nie znajduje się w tablicy stron procesu, to oznacza to, że proces próbuje sięgnąć do nieistniejącej strony lub do strony, która do niego nie należy. Równie łatwo jest sprawdzić, czy przemieszczenie w adresie logicznym jest prawidłowe: nie powinno ono przekraczać rozmiaru pojedynczej strony. Można również wprowadzić dodatkową ochronę. W tablicy stron przy każdej pozycji można umieścić trzy bity, które będą określały rodzaj dostępu do określonej strony: odczyt, zapis, wykonanie (w notacji uniksowej: rwx). Dzięki odpowiednim kombinacją wartości tych bitów można uzyskać różne formy kontroli dostępu do stron, np. tylko odczyt i zapis. Naruszenie ochrony gwarantowanej przez te bity jest wykrywane przez sprzęt i obsługiwane na zasadzie wyjątku przez system operacyjny. Do ochrony samej tablicy stosowany jest niekiedy *rejestr długości tablicy stron*, który również określa, które elementy w tablicy są używane.

Segmentacja

Technika stronicowania jest przezroczysta dla programistów piszących programy w językach wysokiego poziomu oraz w językach asemblerowych. Oznacza to, że programy te tworzone są w ten sam sposób jak dla komputerów, które są pozbawione możliwości stronicowania. Pamięć logiczna procesów jest dzielona na równe obszary, niezależnie od tego czy zawierają one dane, czy kod. W zarządzaniu pamięcią opartym na *segmentacji* (ang. segmentation) pamięć logiczna jest dzielona na obszary o różnej wielkości, które zawierają logicznie odrębne fragmenty kodu. W osobnym segmencie mogą być zamknięte rozkazy programu, w innym mogą być zamknięte dane, a w jeszcze innym stos. Tę segmentację można posunąć jeszcze dalej: w osobnych segmentach można zamykać np.: stałe programu lub poszczególne struktury danych. Segmentacja jest więc bliższa wyobrażeniu programisty na temat pamięci procesu, niż stronicowanie. Programista tworzący oprogramowanie w asemblerze może zazwyczaj określić do jakich segmentów zakwalifikować poszczególne fragmenty programu i ile tych segmentów będzie. W przypadku programisty piszącego w języku wysokiego poziomu ta czynność jest wykonywana automatycznie i niejawnie przez kompilator. Każdy segment ma swój unikalny numer. Podobnie jak w przypadku stronicowania do translacji adresów potrzebna jest tablica nazywana tutaj *tablicą segmentów*. Prosty schemat segmentacji stosuje część procesorów Intela, które wymuszają na programiście podział procesu na trzy segmenty: kodu, danych i stosu. Tablica segmentów jest w ich przypadku realizowana za pomocą rejestrów CS, DS i SS. Segmentacja jest pozbawiona fragmentacji wewnętrznej, ale obarczona jest fragmentacją zewnętrzną. Ponieważ rozmiary segmentów nie są na ogół duże, to nie jest ona tak dotkliwa jak w przypadku przydziału obszarów ciągłych pamięci. Ponadto, ponieważ segmentacja podobnie jak stronicowanie jest formą dynamicznego wiązania adresów, to można do eliminacji fragmentacji zewnętrznej zastosować technikę upakowania.

Translacja adresów



Każdy adres logiczny składa się z numeru segmentu s i z przesunięcia względem początku tego segmentu d. Translacja adresów odbywa się według schematu przedstawionego obok. Numer segmentu z adresu logicznego traktowany jest jako indeks w tablicy segmentów. Każdy element tej tablicy zawiera dwie wartości: wielkość segmentu (granica) oraz adres bazowy segmentu (baza). Jeśli przesunięcie w adresie logicznym jest większe niż wielkość segmentu, to generowany jest wyjątek adresowania. Jeżeli nie, to przesunięcie dodawane jest do adresu bazowego segmentu i wykonywane jest odwołanie do pamięci fizycznej. Translacja adresów w segmentacji wraz ze sprawdzeniem ich poprawności wykonywana jest sprzętowo.

Tablica segmentacji

Podobnie jak w przypadku stronicowania dostęp do tablicy segmentów może zostać przyspieszony za pomocą rejestrów asocjacyjnych. Ponieważ każdy proces posiada własną tablicę segmentów, to konieczny jest również *rejestr bazowy tablicy segmentów* oraz *rejestr długości tablicy segmentów*. Pełnią one tę samą rolę co analogiczne rejestry w stronicowaniu.

Ochrona i współdzielenie segmentów

Kontrola poprawności przesunięcia w adresie logicznym została już opisana. Numer segmentu również podlega sprawdzeniu. Jeśli nie istnieje element tablic wskazywany przez ten numer, to oznacza, że proces odwołuje się do nieistniejącego segmentu lub do segmentu, który nie należy do niego. Taki proces należy zakończyć. Podobnie jak w przypadku stronicowania w tablicy segmentów można umieścić dodatkowe informacje o ochronie poszczególnych segmentów. Mogą to być informacje o trybie dostępu do zawartości tych segmentów (odczyt, zapis, wykonanie). Segmenty mogą być współdzielone, ale należy zachować ostrożność pozwalając na użytkowanie jednego segmentu przez wiele procesów. Jeśli w tym segmencie są umieszczone odwołania do innych segmentów, to powinny mieć te segmenty takie same numery dla wszystkich procesów. Schemat ten się komplikuje, jeśli procesor posiada pośredni tryb adresowania pozwalający na wielokrotne odwołania do różnych segmentów.

Segmentacja stronicowana

Oba opisane wcześniej mechanizmy zarządzania pamięcią można połączyć (oba są również realizacjami koncepcji dynamicznego przydziału pamięci). Z tego połączenia powstaje *segmentacja stronicowana*. Z jednej strony posiada ona tę zaletę stronicowania, że nie jest obarczona fragmentacją zewnętrzną, z drugiej strony pozwala lepiej dopasować się do logicznej struktury kodu procesu co jest zaletą segmentacji. To rozwiązanie stosował system Multics. Adres logiczny w segmentacji stronicowanej ma tę samą strukturę, co w zwykłej segmentacji. Z tablicy segmentów nie jest jednak odczytywany adres bazowy segmentu, ale adres początku tablicy stron tego segmentu. Przesunięcie w segmencie podzielone jest na dwie części: numer strony oraz przesunięcie na tej stronie. Translacja adresu w segmentacji stronicowanej jest więc skomplikowaną czynnością. Dodatkowo utrudnia ją fakt, że tablica segmentów, ze względu na duże rozmiary jest również stronicowana. Ten system zarządzania pamięcią ze względu na swój stopień skomplikowania jest rzadko stosowany.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!