Programowanie Współbieżne

Komunikacja między procesowa IPC
Semafory

Semafory

Semafory są pojęciem pierwotnym w zagadnieniach synchronizacji. Nie służą do wymiany informacji tylko do synchronizowania dostępu do zasobów

Semafory

Przykład operacji semaforowej P(S) w pseudo kodzie:

```
for (;;)
   {
   if (semafor > 0){
      semafor--;
      break;}
   }
```

Tu niestety nie ma zagwarantowanej niepodzielności.

Semafory

Jądro utrzymuje pewną strukturę informacji dla każdego zbioru semaforów w systemie (przykład struktury).

```
#include <sys/types.h>
#include <sys/ipc.h> /* tu definicja struktury
ipc_perm */
struct semid_ds
   {
   struct ipc_perm sem_perm;
   struct sem *sem_base; /* wskaźnik do pierwszego
semafora w zbiorze */
   ushort sem_nsems; /* liczba semaforów */
   time_t sem_otime; /* czas ostatniej operacji */
   time_t sem_ctime; /* czas ostatniej zmiany */
   };
```

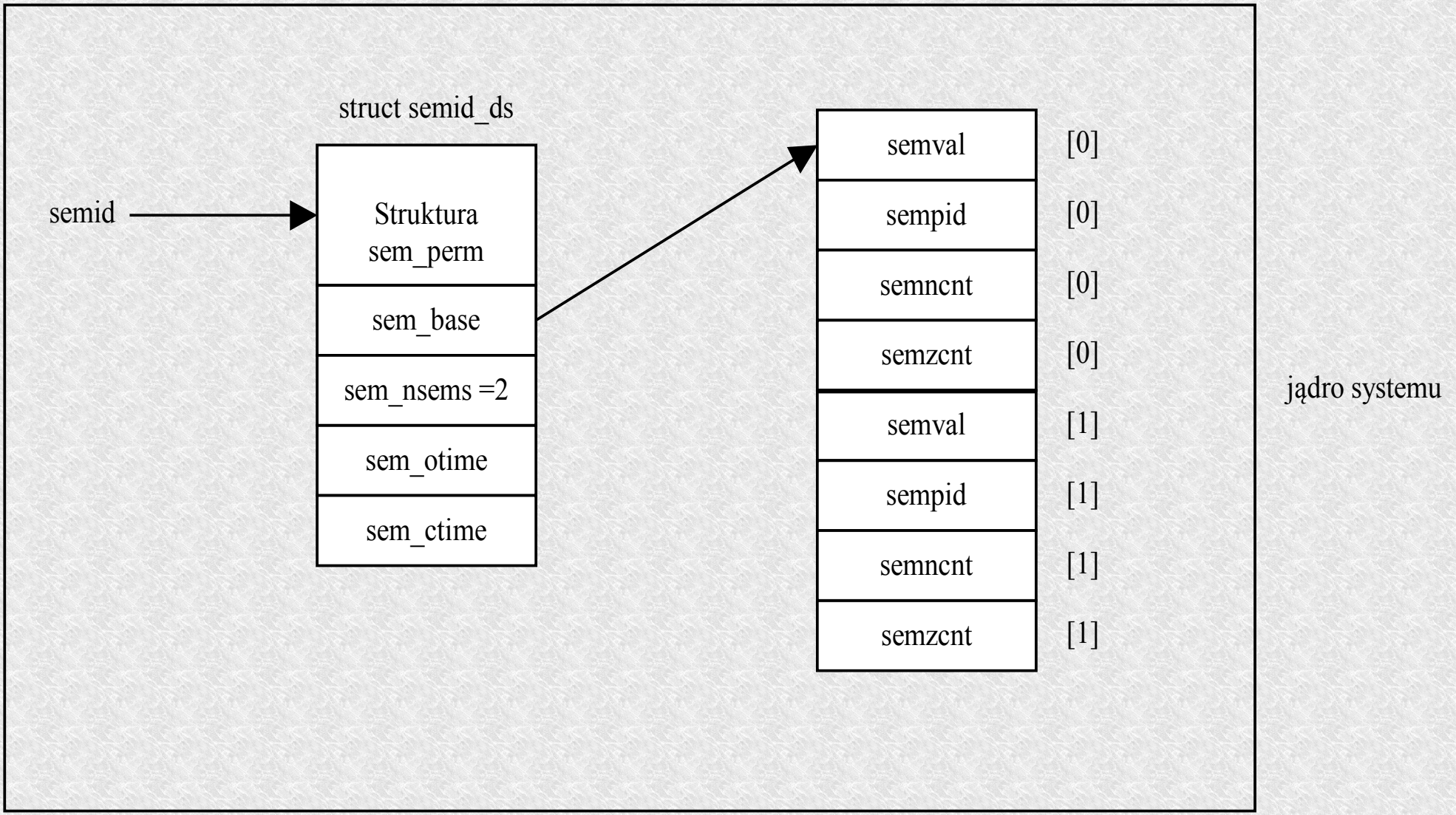
Semafory

sem jest wewnętrzną strukturą danych

```
struct sem {
   ushort semval; /* nieujemna wartość semafora */
   short sempid; /* identyfikator procesu dla
ostatniej operacji */
   ushort semncnt /* liczba oczekujących wartości
semafora > wartości bieżącej */
   ushort semzcnt; /* liczba czekających wartości
semafora = 0 */
```

Semafory

Przykład struktury 2 elementowego semafora



Semafory

Do tworzenia lub otwierania semafora służy funkcja

int semget (key_t key, int nsems, int semflag);

- zwraca identyfikator semafora lub –1
- *nsem* – ile semaforów w zbiorze, jeżeli nie tworzymy tylko otwieramy już dany zbiór semaforów to można dać tu 0
- w utworzonym zbiorze semaforów nie można zmienić ich liczby
- *semflag* – jest kombinacją następujących stałych symbolicznych:

Semafory

Wartość liczbowa	stała symboliczna	znaczenie
0400	*SEM_R*	czytanie przez właściciela
0200	*SEM_A*	zmienianie przez właściciela
0040	SEM_R >> 3	czytanie przez grupę
0020	*SEM_A* >> 3	zmienianie przez grupę
0004	*SEM_R* >> 6	czytanie przez innych
0002	*SEM_A* >> 6	zmienianie przez innych
1000	*IPC_CREAT*	
2000	*IPC_EXCL*	

Semafory

Do wykonywania operacji na semaforach służy funkcja

*int semop(int semid, struct sembuf *opstr, unsigned int nops);*

- zwraca 0 jeśli się powiedzie lub –1 w przypadku błędu
-
- *semid* – identyfikator semafora
- *nops* – liczba elementów w tablicy struktur *sembuf* na którą wskazuje *opstr*
- *opstr* - wskazuje na tablicę następujących struktur:

```
struct sembuf {
   ushort sem_num; /*numer semafora */
   short sem_op; /* operacja na semaforze */
   short sem_flag; /* znacznik operacji */
};
```

Semafory

- każdy element tej tablicy określa operację na wartości jednego semafora ze zbioru semaforów
  - *sem_num* – określa który semafor (licząc od 0)
  - *sem_op*
    - *>0 wartość tą dodaj do bieżącego semafora (uwolnij zasoby) operacja V(s)*
    - *= 0 proces wywołujący funkcję semop chce czekać , aż wartością semafora stanie się zero.*
    - *<0 proces wywołujący czeka aż wartość semafora stanie się większa niż (lub taka sama jak) wartość bezwzględna tego pola. Następnie zostaną zsumowane, czyli przydział zasobów operacja P(s). Np. s=1+(-1). s=0 semafor opuszczony.*
  - *sem_flg – ma kilka opcji np. cofanie zmian wykonanych przez proces na tym semaforze jeżeli proces „padnie”*
    - *IPC_NOWAIT na sem_flg informuje system że nie chcemy czekać, aż operacja będzie ukończona.*

Semafory

Do operacji sterujących na semaforze służy funkcja

```
int semctl(int semid, int semnum, int cmd, union semun
arg);
```

- semun – jest zbudowana następująco

```
union semun {
int val; /* używane tylko dla SETVAL */
struct semid_ds. *buff; /* używane dla IPC_STAT oraz
IPC_SET */
ushort *array; /* używane dla GETVAL oraz SETVAL */
} arg;
```

- cmd – polecenie
  - IPC_RMID – usunięcie semafora
  - GETVAL – pobranie wartości semafora, funkcja zwróci jego wartość
  - SETVAL – nadanie wartości semoforowi val w unii semun;
- semnum – którego semafora dotyczy

Semafory

Zajmowanie zasobów przy użyciu semaforów

- semafory można traktować jako mechanizm synchronizacyjny
- załóżmy, że wartość semafora 1 będzie oznaczać zasób zajęty a 0 zasób wolny
- założenie to jest trochę odwrotne do naszej logiki semaforów ale w niektórych systemach nie da się nadać wartości początkowej semaforowi innej niż 0

Semafory

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#define SEMKEY 123456L /* wartosc klucza dla funkcji
systemowej semget() */
#define PERMS 0666

static struct sembuf op_lock[2] = {
   0,0,0, /* czekaj, aż semafor nr 0 stanie się zerem */
   0,1,0 /* następnie zwiększ ten semafor  1*/
};

static struct sembuf op_unlock[1] = {
   0, -1, IPC_NOWAIT /* zmniejsz semafor nr 0 o 1 bez
czekania bo to zwolnienie zasobów */
};

int semid = -1; /* identyfikator semafora */
```

Semafory

```
my_lock()
{
if (semid <0)
    {
    if ((semid = semget(SEMKEY, 1, IPC_CREAT| PERMS)) < 0)
       perror(„blad tworzenia semafora”);
    }
if (semop(semid, &op_lock[0], 2) < 0)
   perror(„blad zajmowania semafora”);
}

my_unlock()
{
if (semop (semid, &op_unlock[0], 1) < 0)
   perror(„blad odblokowywania”);
}
```

Semafory

Co w przypadku gdy któryś z procesów „padnie”?

- Można „uzbroić” proces zajmujący semafor, w obsługę większości sygnałów jakie mogą nadejść i w każdym wywołaniu elegancko zwalniać semafor. Nie zda się to niestety do *SIG_KILL*

Semafory

- funkcja *my_lock* może określić znacznik *IPC_NOWAIT* w pierwszej operacji w tablicy *op_lock. J*eżeli funkcja operacji zwróci –1 a *errno* = *EAGAIN* to proces może wywołać semctl i zbadać wartość pola *sem_ctime* struktury *semid_ds*. dla tego semfaora, jeżeli okaże się że upłynął z góry założony czas (np. 10s) od ostatniej zmiany to proces może zająć zasób przyjąwszy że inny proces już go nie potrzebuje, a zapomniał/nie zdołał zwolnić

Wadą jest to że trzeba ciągle wywoływać dodatkową funkcje gdy zasób jest zajęty i trzeba przyjąć jakiś *TIMEOUT*

Semafory

- Trzecie najlepsze rozwiązanie jest takie że powiadamiamy jądro podczas zajmowania zasobu, że jeżeli proces zostanie zakończony przed uwolnieniem tego zasobu, to jądro ma go uwolnić.

Semafory

Wartość nastawna semafora – dla każdej wartośc semafora w systemie można określić drugą, związaną z nią wartość. Tzw (*Smaphore Adjustment Value*).

- Gdy określa się wartość początkową semafora wówczas ustanawia się wartość nastawiającą dany semafor równą 0.
- Dla każdej operacji użytej w wywołaniu funkcji semop i mającej ustawiony znacznik *SEM_UNDO*, jeżeli zwiększy się wartość semafora, to również o tyle samo będzie zmniejszona wartość nastawiająca ten semafor.

•Kiedy proces zakończy się wtedy jądro użyje automatycznie wszystkich wartości nastawnych dla tego procesu. (zostanie zsumowana z wartością semafora).

Semafory

Przykład do poprzedniego zajmowania:

```
static struct sembuf op_lock[2] = {
   0,0,0,
/* czekaj, aż semafor nr 0 stanie się zerem */
   0,1,SEM_UNDO
/* następnie zwiększ ten semafor  1*/
};

static struct sembuf op_unlock[1] = {
   0, -1, IPC_NOWAIT | SEM_UNDO
   /* zmniejsz semafor nr 0 o 1 bez czekania bo to
zwolnienie zasobów */
};
```

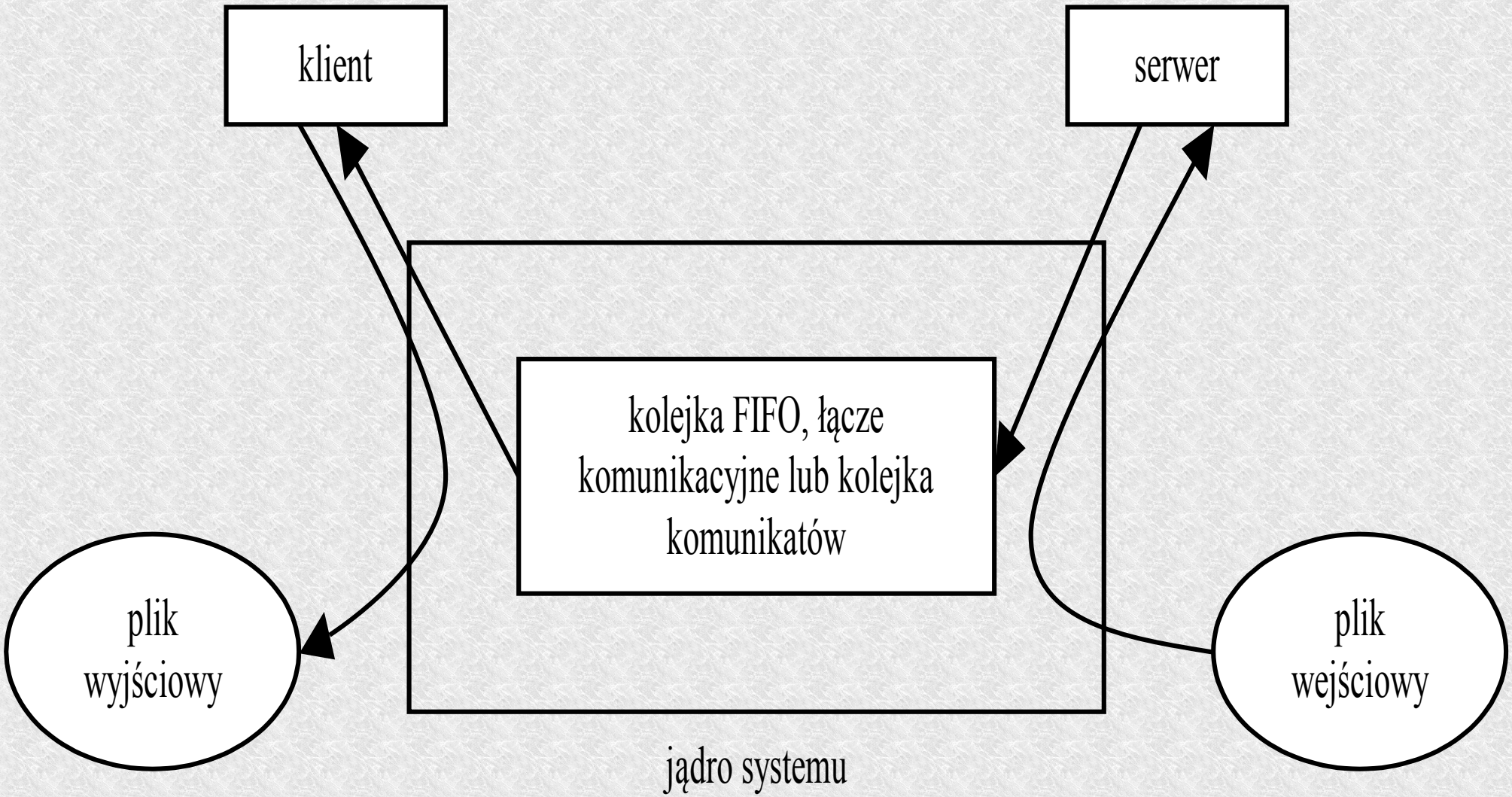
Trzeba też pamiętać by ostatni proces używający semafora usunął go z systemu

Pamięć Dzielona

Jeszcze raz program (klient-serwer) kopiujący pliki

- Serwer czyta plik, zwykle jądro kopiuje z dysku do jakiegoś bufora
- z tego bufora trafia do bufora naszego serwera określonego jako drugi argument funkcji read
- Serwer zapisuje te dane do kolejki FIFO, łącza nienazwanego czy kolejki komunikatów, znowu jest kopiowanie z bufora użytkownika do jądra
- Klient czyta dane z kanału IPC to wymaga kopiowania z bufora w jądrze do bufora klienta
- w końcu kopiuje z bufora klienta do bufora wyjściowego (write)
- a z bufora wyjściowego np. na ekran.

Pamięć Dzielona



Pamięć Dzielona

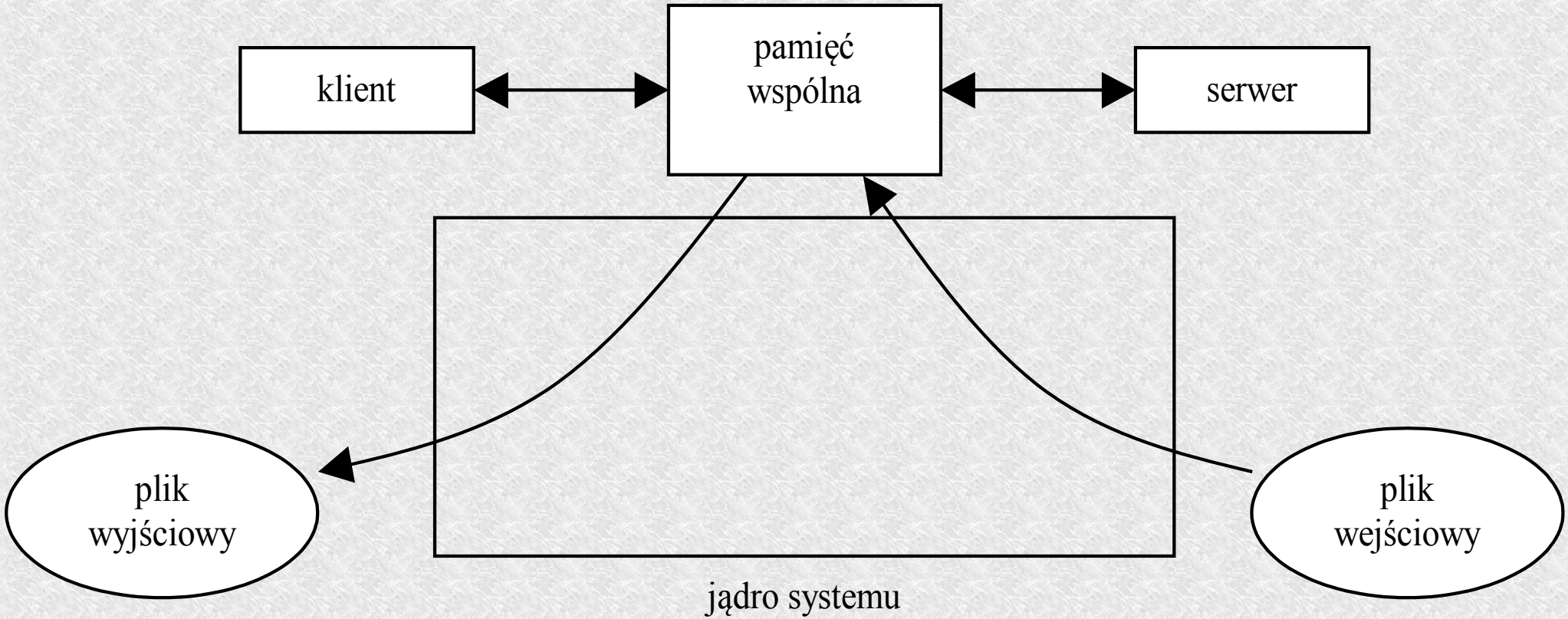
- Pamięć wspólna pozwala ominąć niedogodność związaną ze zbyt dużą liczbą kopiowań danych pozwalając korzystać z tego samego segmentu pamięci dwóm procesom lub więcej.
- Współdzielenie pamięci jest podobne do korzystania ze wspólnego pliku. Trzeba zastosować dodatkowe mechanizmy synchronizacji, np. semafory.

Pamięć Dzielona

Algorytm działania będzie następujący:

- Serwer uzyskuje dostęp do segmentu pamięci wspólnej używając semafora
- Serwer czyta do pamięci wspólnej
- Po zakończeniu czytania serwer zawiadamia klienta posługując się semaforem, że dane już są gotowe do odebrania z pamięci
- Klient czyta z pamięci wspólnej i zapisuje do pliku wynikowego

Pamięć Dzielona



Pamięć Dzielona

dla każdego segmentu pamięci wspólnej jądro systemu utrzymuje następującą strukturę z informacjami:

```
struct shmid_ds {
struct ipc_perm shm_perm; / * struktura praw dostępu
do operacji */
int shm_segsz; /*rozmiar segmentu */
struct XXX shm_YYY; /* informacje zależne od
realizacji */
ushort shm_lpid; /*identyfikator procesu dla ostatniej
operacji */
ushort shm_cpid; /* twórca identyfikatora procesu */
ushort shm_nattch; /* dołączony numer bieżący */
ushort shm_cnattch; /* dołączony numer w pamięci
wewnętrznej */
time_t shm_atime; /* czas ostatniego dołączenia */
time_t shm_dtime; /* czas ostatniego odłączenia */
time_t shm_ctime; /* czas ostatniej zmiany */
};
```

Pamięć Dzielona

do tworzenia segmentu pamięci wspólnej służy funkcja:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget (key_t key, int size , int
shmflag);
```

- zwraca identyfikator pamięci lub –1 w razie błędu
- *key* klucz utworzony przez *ftok* lub wymyślony przez nas
- size – rozmiar pamięci w bajtach
- shmflag – kombinacja znaczników

Pamięć Dzielona

Wartość liczbowa	stała symboliczna	znaczenie
0400	*SEM_R*	czytanie przez właściciela
0200	*SEM_A*	zmienianie przez właściciela
0040	SEM_R >> 3	czytanie przez grupę
0020	*SEM_A* >> 3	zmienianie przez grupę
0004	*SEM_R* >> 6	czytanie przez innych
0002	*SEM_A* >> 6	zmienianie przez innych
1000	*IPC_CREAT*	
2000	*IPC_EXCL*	

Pamięć Dzielona

Dołączenie segmentu pamięci:

```
char *shmat(int shmid, char *shmaddr, int shmflag);
```

- przekazuje adres początkowy segmentu pamięci wspólnej
- *shmid* – identyfikator pamięci zwrócony przez shmget
- *shmflag* może mieć znacznik *(SHM_RDONLY)*
- adres jest ustalany według następujących zasad:

Pamięć Dzielona

- jeżeli *shmaddr* = 0 to system sam wybiera adres (sprawdza się w większości zastosowań.
- jeżeli jest != 0 to przekazywany adres zależy od tego czy ustalony jest znacznik *SHM_RND*
    - jeżeli nie jest ustawiony to segment pamięci wspólnej będzie podłączony od adresu określonego przez argument shmaddr
    - jeżeli jest ustawiony to zacznie się od adresu zaokrąglonego w dół o wartość stałej *SHMLBA (Lower Boundary Address)*

Pamięć Dzielona

odłączenie pamięci wspólnej dokonuje się za pomocą

```
int shmdt(char *shmaddr);
```

funkcja ta nie usuwa segmentu pamięci !

Pamięć Dzielona

By usunąć segment pamięci dzielonej trzeba użyć

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

z argumentem cmd jako IPC_RMID