

„Materiały wprowadzające”

dr inż. Arkadiusz Chrobot

24 lutego 2024

Spis treści

Wprowadzenie	1
1. Polecenie ssh	1
2. Polecenie scp	1
3. Linux Cross Reference	2

Wprowadzenie

W materiałach wstępnych zawarte są krótkie opisy narzędzi, których znajomość będzie niezbędna do sprawnego wykonania zadań w ramach laboratoriów z Systemów Operacyjnych 2. Każdemu zespołowi zostanie przypisana maszyna wirtualna, którą trzeba będzie obsługiwać przy pomocy konsoli, dlatego **należy przypomnieć sobie podstawowe polecenia powłoki systemowej**. Te polecenia nie będą opisane w tych materiałach. W rozdziale 1 przedstawiony jest sposób posługiwania się poleceniem `ssh` (ang. *Secure Shell*), w rozdziale 2 opisano polecenie `scp` (ang. *Secure Copy*), a rozdział 3 poświęcony jest użytkownikowi serwisu Linux Cross Reference.

1. Polecenie ssh

Polecenie `ssh` służy do zdalnej pracy z komputerem. Praca ta odbywa się za pomocą powłoki systemowej (konsoli), a komunikacja między komputerem lokalnym i zdalnym jest szyfrowana. W przypadku zajęć laboratoryjnych tym zdalnym komputerem będzie maszyna wirtualna. Aby się z nią połączyć należy wydać na lokalnym komputerze, w terminalu lub konsoli, następujące polecenie:

```
ssh so2@81.26.20.32 -p X
```

W tym poleceniu `so2` to nazwa użytkownika, `81.26.20.32` to adres IP serwera maszyn wirtualnych, a `X` oznacza numer portu. Każdy zespół będzie miał przypisaną inną maszynę, z którą będzie się komunikował na określonym porcie. Numer tego portu będzie przekazany na pierwszych lub drugich zajęciach. Po nawiązaniu połączenia przez polecenie `ssh` zdalny system zapyta o hasło. Ono także zostanie przekazane w trakcie laboratorium. Podczas pisania hasła na ekranie nie będą się pojawiały żadne znaki, do czasu naciśnięcia klawisza `Enter`. **Należy zaznaczyć, że z maszyną wirtualną można połączyć się tylko i wyłącznie z wewnętrznej sieci Katedry Systemów Informatycznych i jedynie wtedy, gdy ta maszyna jest uruchomiona.**

Opisane polecenie pozwala zalogować się na konto użytkownika `so2`, które nie jest kontem uprzywilejowanym. Aby móc załadować lub usunąć moduły do jądra, co jest konieczne do realizacji zadań z instrukcji, należy skorzystać z konta użytkownika `root`. Można to zrobić następującym poleceniem:

```
su -
```

Po jego wydaniu system zapyta o hasło dla użytkownika `root`. Ono również zostanie przekazane przez osobę prowadzącą na zajęciach. Po podaniu prawidłowego hasła uzyskuje się dostęp do konta pozwalającego na administrację systemem, w tym wykonywanie wspomnianych operacji.

Inny sposób zalogowania się na konto użytkownika `root` polega na wydaniu polecenia:

```
sudo su -
```

W tym przypadku system zapyta o hasło użytkownika `so2`, a nie `root`. Jeśli będzie ono prawidłowe, to tak jak poprzednio otrzymuje się dostęp do powłoki systemowej tego ostatniego użytkownika.

Polecenie `sudo` ma też inne zastosowania. Ogólnie służy do uruchamiania poleceń uprzywilejowanych z poziomu użytkownika nieuprzywilejowanego. Nie mogą to być jednak dowolne polecenia. Muszą one być wcześniej określone przez administratora systemu dla tego użytkownika.

2. Polecenie scp

Polecenie `scp` należy do tego samego pakietu, co `ssh`. Umożliwia ono kopiowanie plików pomiędzy komputerem lokalnym i zdalnym przy użyciu szyfrowanej komunikacji. W przypadku laboratoriów, aby skopiować plik o nazwie `source.c` znajdujący się w bieżącym katalogu na komputerze lokalnym, do

katalogu domowego użytkownika `so2` na maszynie wirtualnej o porcie `X` należy wydać następujące polecenie:

```
scp -P X source.c so2@81.26.20.32:~
```

Proszę zwrócić uwagę na znak dwukropka po adresie IP, po którym występuje nazwa katalogu. W tym wypadku jest to katalog domowy użytkownika, który w Linuksie oznaczany jest znakiem tyldy (`~`). Jeśli miałby to być podkatalog katalogu domowego, np. o nazwie `lab1`, to należałoby dokonać następującej zmiany w tym poleceniu:

```
scp -P X source.c so2@81.26.20.32:~/lab1
```

Podobnie jak w przypadku polecenia `ssh`, zdalny system zapyta o hasło użytkownika `so2`.

Analogicznie można skopiować pliki w odwrotnym kierunku, czyli np. aby skopiować plik `Makefile` znajdujący się w katalogu domowym użytkownika `so2`, na maszynie wirtualnej, do bieżącego katalogu na komputerze lokalnym, należy wydać następujące polecenie:

```
scp -P X so2@81.26.20.32:~/Makefile .
```

Kropka na końcu polecenia oznacza katalogu bieżący. Proszę zwrócić uwagę, że kolejność argumentów w `scp` jest taka sama, jak w zwykłym poleceniu `cp`, które kopiuje pliki lokalnie.

Aby skopiować cały katalog do zdalnego systemu, można użyć jednego z dwóch sposobów. W przykładach założono, że katalog do skopiowania nazywa się `sources`, jest podkatalogiem bieżącego katalogu i należy go umieścić w katalogu domowym użytkownika `so2` na maszynie wirtualnej.

Pierwszy sposób polega na użyciu opcji `-r` polecenia `scp`, która powoduje rekurencyjne skopiowanie katalogu:

```
scp -P X -r sources so2@81.26.20.32:~
```

W drugim sposobie najpierw należy ten katalog skompresować np. za pomocą polecenia `tar`. Można to zrobić następująco:

```
tar jcvf sources.tar.bz2 sources
```

Plik o nazwie `sources.tar.bz2` będzie skompresowanym archiwum katalogu `sources`. Po skopiowaniu go do zdalnego komputera można go rozpakować poleceniem:

```
tar xvf sources.tar.bz2
```

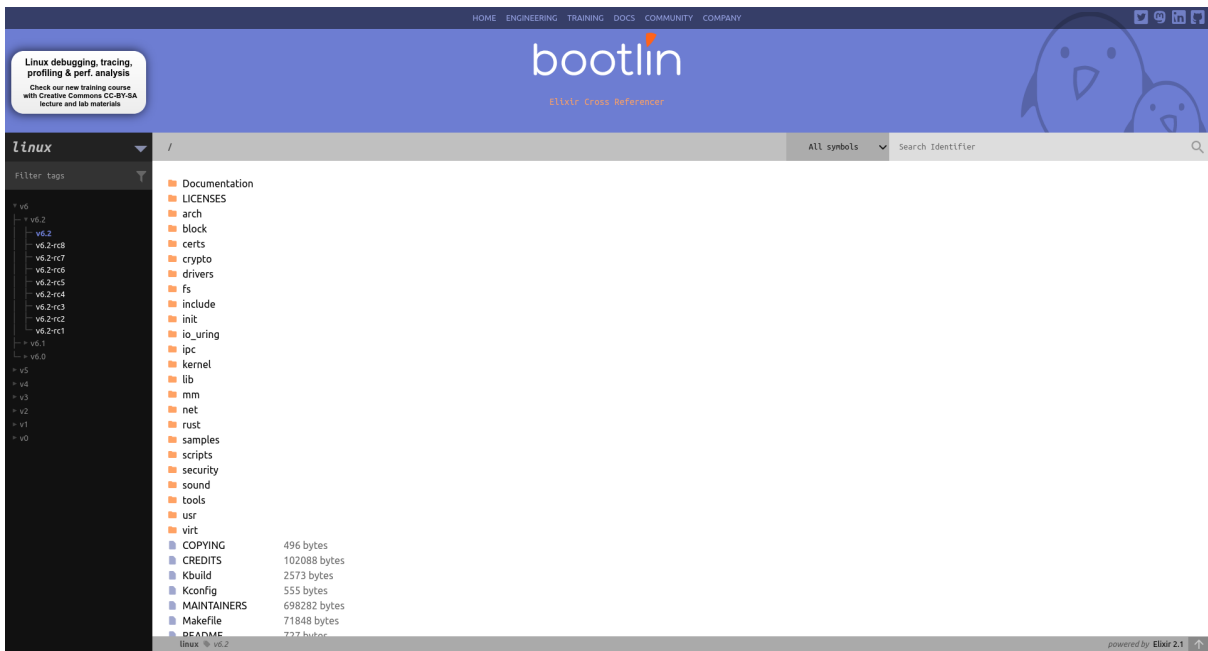
3. Linux Cross Reference

Istnieje dokumentacja do kodu źródłowego jądra Linuksa w postaci stron podręcznika `man`, ale niestety jest ona bardzo ograniczona. Najlepszym sposobem na zbadanie, co dany fragment kodu robi i jak jest zbudowany, jest przeglądanie źródeł jądra, które są bardzo obszerne. Na szczęście istnieje narzędzie, które potrafi wygenerować strony HTML pozwalające łatwo nawigować po tym kodzie. Tak wygenerowane strony dostępne są, między innymi, pod adresem <https://elixir.bootlin.com/linux/latest/source>.

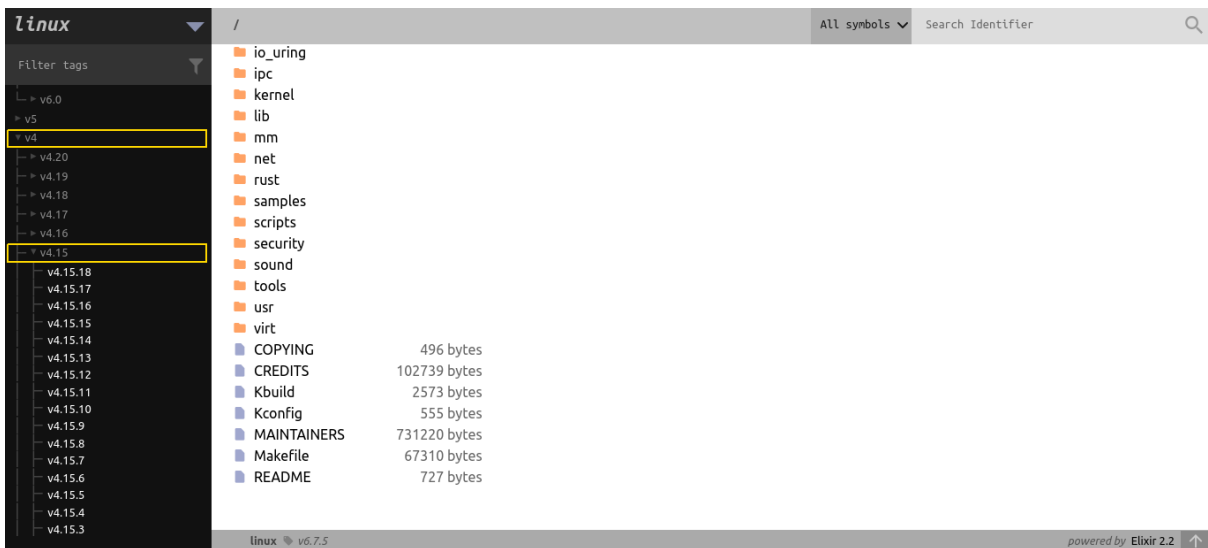
Sposób korzystania z nich zostanie opisany na przykładzie wyszukiwania informacji o definicji funkcji `schedule()`¹. Po wprowadzeniu wymienionego adresu do przeglądarki i naciśnięciu klawisza `Enter`² pojawi się strona, której fragment jest widoczny na rysunku 1. Na tej stronie należy wybrać wersję 4.15 źródeł jądra, w sposób pokazany na rysunkach 2 i 3. Po wybraniu wersji kodu źródłowego jądra należy w polu *Search Identifier*, wprowadzić nazwę `schedule`, a następnie kliknąć ikonę lupy lub nacisnąć klawisz `Enter` (rysunek 4). Strona z wynikami jest podzielona na dwie części. Pierwsza część, zaznaczona na rysunku 5 na żółto, zawiera odnośniki do wierszy w plikach, w których jest umieszczony prototyp lub definicja szukanej funkcji. Druga część, której fragment jest oznaczony kolorem zielonym, to lista odnośników do wierszy plików, w których ta funkcja jest używana. Po kliknięciu w odnośnik w sekcji *Defined in 1 files as a prototype*: użytkownik zostanie przeniesiony do strony z deklaracją funkcji. Jej prototyp (nagłówek) został oznaczony na żółto na rysunku 6. Po kliknięciu odnośnika w sekcji *Defined in 1 file as a function*: (na rysunku 5 zaznaczona na zielono) użytkownik zostanie przeniesiony na stronę z definicją tej funkcji, która została zaznaczona na żółto na rysunku 7.

¹Ta funkcja jest implementacją planisty procesora.

²Alternatywnie można kliknąć odnośnik na stronie przedmiotu.



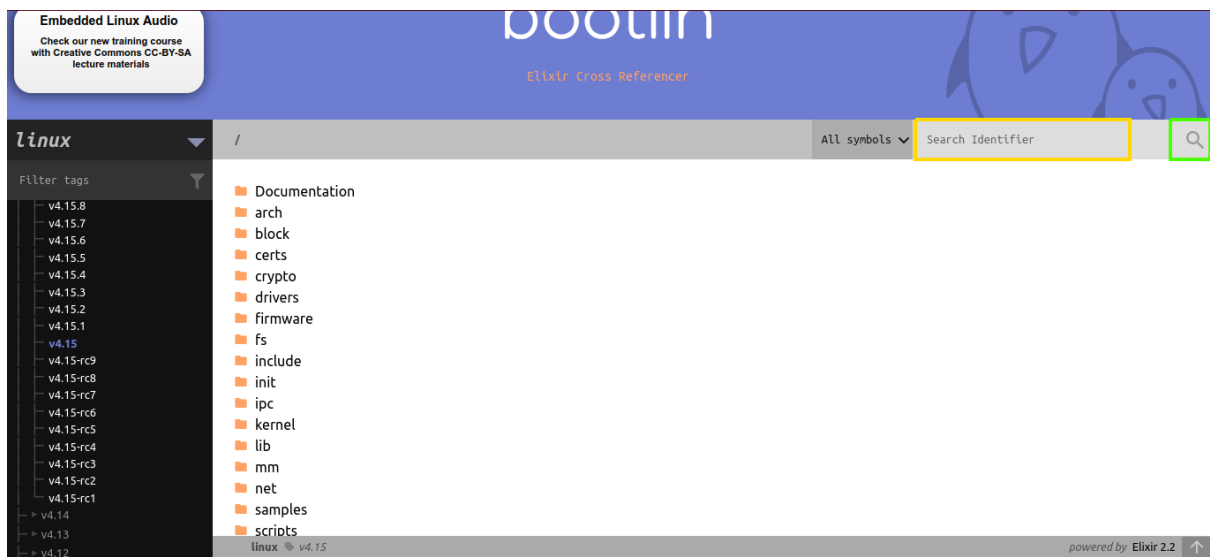
Rysunek 1: Strona główna serwisu Linux Cross Reference



Rysunek 2: Menu wyboru wersji źródeł jądra



Rysunek 3: Szczegółowe menu wyboru wersji źródeł jądra



Rysunek 4: Wyszukiwanie nazwy

The screenshot shows the bootlin website interface. At the top, there is a navigation bar with links for HOME, ENGINEERING, TRAINING, DOCS, COMMUNITY, and COMPANY. The bootlin logo is prominently displayed. Below the navigation bar, there is a search bar and a filter menu. The search results for the symbol 'schedule' are shown, categorized into three sections:

- Defined in 1 files as a prototype:**
 - include/linux/sched.h, line 178 (as a prototype)
- Defined in 11 files as a member:**
 - drivers/gpu/drm/915/intel_ringbuffer.h, line 407 (as a member)
 - drivers/net/usb/r8152.c, line 715 (as a member)
 - drivers/net/wireless/ath/wcn36xx/hal.h, line 2454 (as a member)
 - drivers/net/wireless/intel/iwlfw/fw/api/scan.h, line 368 (as a member)
 - drivers/net/wireless/intel/iwlfw/fw/api/scan.h, line 623 (as a member)
 - drivers/scsi/hcr53c8xx.c, line 1348 (as a member)
 - drivers/usb/host/isp116x.h, line 324 (as a member)
 - drivers/usb/host/isp1362.h, line 454 (as a member)
 - drivers/usb/host/s1811.h, line 189 (as a member)
 - include/net/p_vs.h, line 718 (as a member)
- Defined in 1 files as a function:**
 - kernel/sched/core.c, line 3427 (as a function)
- Referenced in 452 files:**
 - arch/alpha/kernel/entry.S, line 536
 - arch/alpha/kernel/signal.c, line 539
 - arch/arc/kernel/entry-compact.S, line 354
 - arch/arc/kernel/entry.S, 2 times
 - arch/arm/kernel/signal.c, line 642
 - arch/arm64/kernel/signal.c, line 918

Rysunek 5: Wyniki wyszukiwania nazwy

The screenshot shows the source code for the 'schedule' function in the kernel. The code is displayed in a dark-themed editor with a sidebar on the left showing a file tree. The function signature is highlighted in yellow:

```

extern void schedule(void);

```

The code includes various headers and defines several structures and macros related to CPU time accounting and scheduling. Key structures include 'prev_cputime', 'task_cputime', and 'task_vtime_state'. The function 'schedule' is defined as a void function that takes no arguments.

Rysunek 6: Deklaracja funkcji schedule()

```

linux / kernel / sched / core.c All symbols Search Identifier
Filter tags
v6
v5
v4.20
v4.19
v4.18
v4.17
v4.16
v4.15
v4.15.18
v4.15.17
v4.15.16
v4.15.15
v4.15.14
v4.15.13
v4.15.12
v4.15.11
v4.15.10
v4.15.9
v4.15.8
v4.15.7
v4.15.6
v4.15.5
v4.15.4
v4.15.3
v4.15.2
v4.15.1
v4.15
v4.15rc9
v4.15rc8
v4.15rc7
v4.15rc6
v4.15rc5
v4.15rc4
v4.15rc3
v4.15rc2
v4.15rc1
v4.14
v4.13
v4.12
v4.11
v4.10
v4.9
v4.8
v4.7
3421 /* make sure to submit it to avoid deadlocks.
3422 */
3423 if (!k_needs_flush_plug(task))
3424 blk_schedule_flush_plug(task);
3425 }
3426
3427 asmlinkage __visible void __sched schedule(void)
3428 {
3429     struct task_struct *task = current;
3430
3431     sched_submit_work(task);
3432     do {
3433         preempt_disable();
3434         __schedule(false);
3435         sched_preempt_enable_no_resched();
3436     } while (need_resched());
3437 EXPORT_SYMBOL(schedule);
3438
3439 /*
3440 * synchronize_rcu_tasks() makes sure that no task is stuck in preempted
3441 * state (have scheduled out non-voluntarily) by making sure that all
3442 * tasks have either left the run queue or have gone into user space.
3443 * As idle tasks do not do either, they must not ever be preempted
3444 * (schedule out non-voluntarily).
3445 *
3446 * schedule_idle() is similar to schedule_preempt_disable() except that it
3447 * never enables preemption because it does not call sched_submit_work().
3448 */
3449 void __sched schedule_idle(void)
3450 {
3451     /*
3452     * As this skips calling sched_submit_work(), which the idle task does
3453     * regardless because that function is a nop when the task is in a
3454     * TASK_RUNNING state, make sure this isn't used somewhere that the
3455     * current task can be in any other state. Note, idle is always in the
3456     * TASK_RUNNING state.
3457     */
3458     WARN_ON_ONCE(current->state);
3459     do {
3460         __schedule(false);
3461     } while (need_resched());
3462 }
3463
3464 #ifdef CONFIG_CONTEXT_TRACKING
3465 asmlinkage __visible void __sched schedule_user(void)
3466 {
3467     /*
3468     * If we come here after a random call to set_need_resched(),
3469     * or we have been woken up remotely but the IPI has not yet arrived,
3470     * we haven't yet exited the RCU idle mode. Do it here manually until
3471     * we find a better solution.
3472     */
3473     /*
3474     * NB: There are buggy callers of this function. Ideally we
3475     * should warn if prev_state != CONTEXT_USER, but that will trigger
3476     */
3477 }
3478 #endif

```

Rysunek 7: Definicja funkcji schedule()