

„Materiały wprowadzające”

dr inż. Arkadiusz Chrobot

22 lutego 2020

Spis treści

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

Wprowadzenie

W tych 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, która 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 opisany jest sposób posługiwania się poleceniem `ssh` (ang. *Secure Shell*), w rozdziale 2 przedstawiono polecenie `scp` (ang. *Secure Copy*), a rozdział 3 poświęcony jest użytkownikowi serwisu Linux Cross Reference.

1. ssh

Polecenie `ssh` służy do zdalnej pracy z komputerem. Praca ta odbywa się za pomocą powłoki (konsoli). Pierwszą czynnością w tej pracy jest zalogowanie się do zdalnego systemu. Najpierw należy się zalogować do serwera maszyn wirtualnych jako użytkownik `so2lab`, wydając polecenie:

```
ssh so2lab@zefir.tu.kielce.pl
```

Zdalny system zapyta o hasło. Należy wprowadzić to, które poda osoba prowadząca laboratorium. Podczas pisania hasła na ekranie nie będą się pojawiały żadne znaki, do czasu naciśnięcia klawisza `Enter`. **Proszę uważnie wprowadzać hasło, bo zbyt duża liczba pomyłek może doprowadzić do czasowego zablokowania konta!!!** Po zalogowaniu się do serwera można zalogować się do maszyny wirtualnej. Maszyny wirtualne, które będą dostępne w ramach laboratoriów będą miały przypisany adres IP postaci `192.168.0.X`, gdzie `X` jest ostatnią liczbą adresu maszyny wirtualnej przypisanej do konkretnego zespołu. Aby się do niej zalogować na konto użytkownika `student` należy wydać następujące polecenie:

```
ssh student@192.168.0.X
```

Kiedy maszyna zapyta o hasło należy je wpisać, pamiętając, że w czasie wpisywania na ekranie nie będą pojawiały się żadne znaki¹.

Aby zalogować się do tej samej maszyny wirtualnej, ale na konto użytkownika `root` należy wydać następujące polecenie:

```
ssh root@192.168.0.X
```

Również w tym przypadku należy podać hasło, kiedy zdalny komputer o nie poprosi.

Istnieje również alternatywny sposób zalogowania się na konto użytkownika `root`. Należy najpierw zalogować się jako użytkownik `student`, a następnie wydać polecenie:

```
su -
```

Po wydaniu tego polecenia system zapyta o hasło dla użytkownika `root`. Należy je podać.

Trzeci sposób zalogowania się na konto użytkownika `root` jest bardzo podobny do drugiego. Pierwszym korkiem również jest zalogowanie do maszyny wirtualnej na konto `student`, ale po wykonaniu tej czynności należy wydać polecenie:

```
sudo su -
```

W tym przypadku system zapyta o hasło użytkownika `student`, a nie `root`.

¹Hasła do poszczególnych kont zostaną podane na zajęciach.

2. scp

Polecenie `scp` należy do tego samego pakietu, co `ssh`. Umożliwia ono kopiowanie plików pomiędzy komputerem lokalnym, a zdalnym. Aby skopiować plik o nazwie `source.c` znajdujący się w bieżącym katalogu na serwerze maszyn wirtualnych, do katalogu domowego użytkownika `root` na maszynie wirtualnej o adresie IP `192.168.0.X` należy wydać następujące polecenie:

```
scp source.c root@192.168.0.X:~
```

Proszę zwrócić uwagę na znak dwukropka po adresie IP, po którym występuje nazwa katalogu. W tym wypadku jest to katalog domowy użytkownika, który oznaczany jest znakiem tyldy (`~`).

Analogicznie można skopiować pliki w odwrotnym kierunku lub między serwerem maszyn wirtualnych, a komputerem z którego jesteśmy zalogowani do tego serwera.

Jeśli chcielibyśmy skopiować cały katalog do zdalnego systemu, to mamy do dyspozycji dwa sposoby. Załóżmy, że ten katalog nazywa się `sources` i jest podkatalogiem bieżącego katalogu.

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

```
scp -r sources root@192.168.0.X:~
```

W drugim sposobie najpierw musimy ten katalog skompresować np. za pomocą polecenie `tar`. Możemy to zrobić następująco:

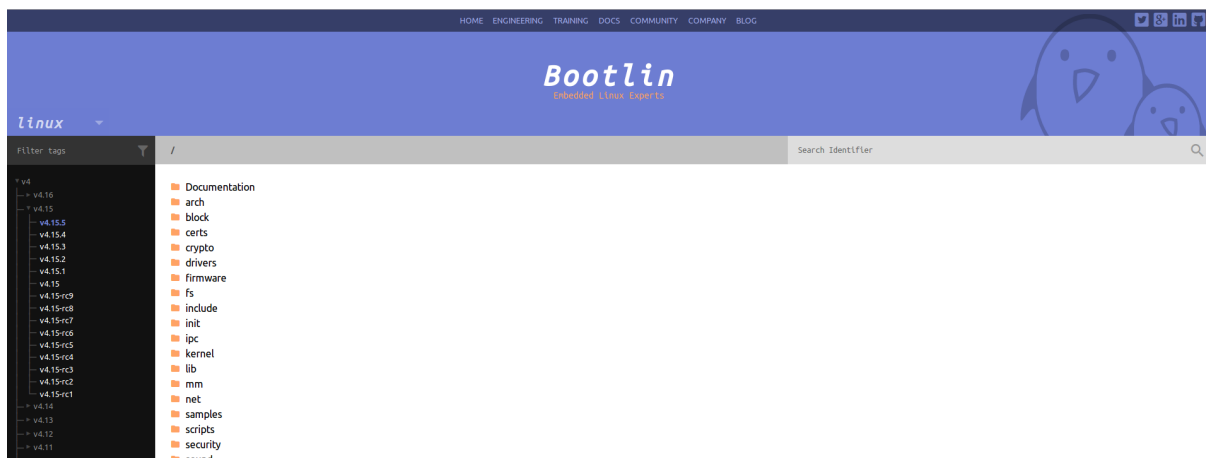
```
tar jcvf sources.tar.bz2 sources,
```

gdzie `sources.tar.bz2` jest nazwą pliku będącego skompresowanym archiwum katalogu `sources`. Po skopiowaniu go do zdalnego komputera możemy 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 opracowane strony dostępne są, między innymi, pod adresem <https://elixir.bootlin.com/linux/latest/source>. Poniżej opisany jest sposób korzystania z tego serwisu na przykładzie wyszukiwania informacji o funkcji `schedule()`². Po wprowadzeniu wspomnianego adresu do przeglądarki i naciśnięciu klawisza `Enter`³ powinniśmy zobaczyć stronę, której fragment jest widoczny na rysunku 1.

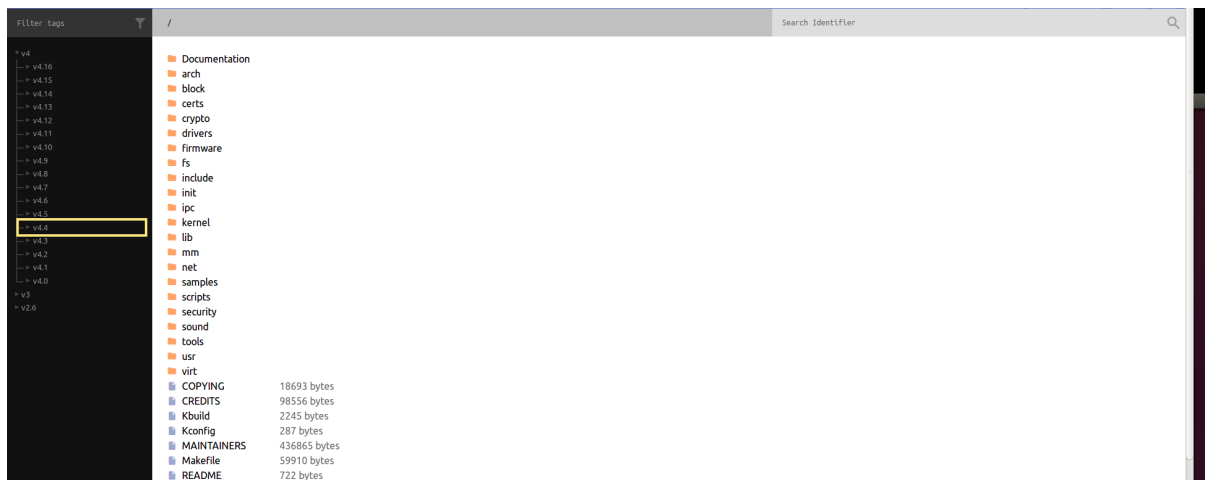


Rysunek 1: Strona główna serwisu Linux Cross Reference

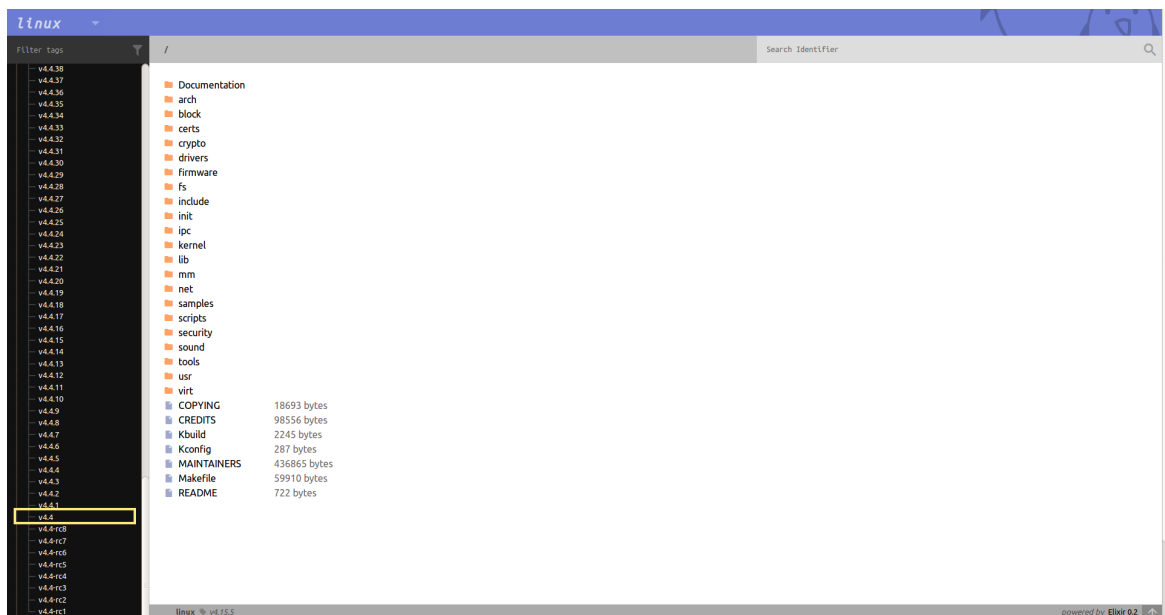
²Ta funkcja jest implementacją planisty procesora.

³Alternatywnie można kliknąć odnośnik na stronie przedmiotu.

Na tej stronie należy wybrać wersję 4.4 źródeł jądra, w sposób pokazany na rysunkach 2 i 3.

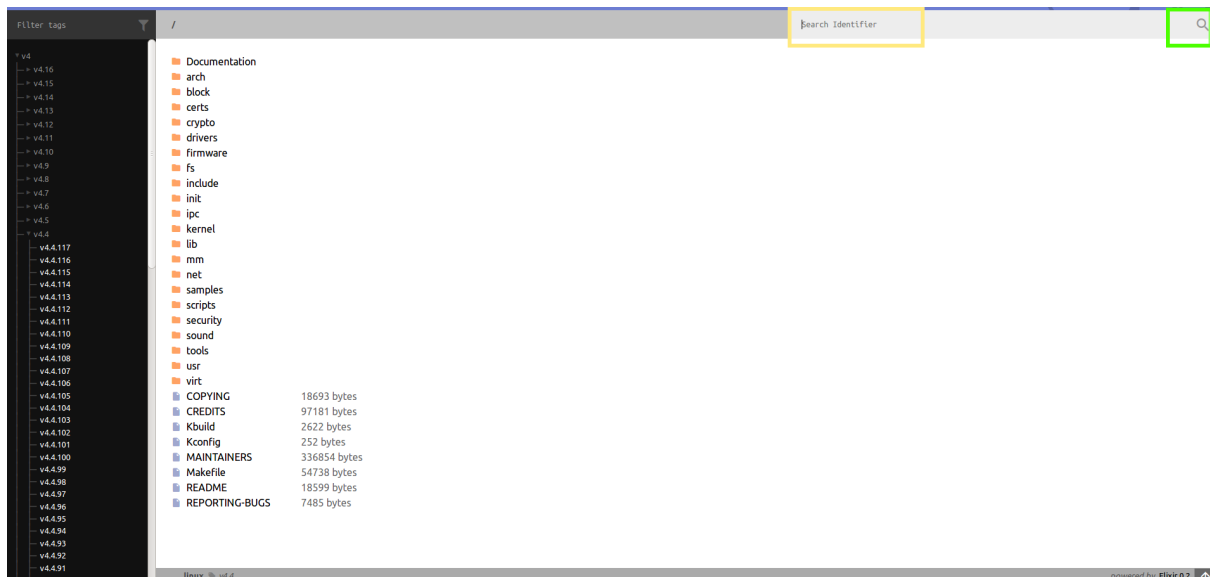


Rysunek 2: Menu wyboru wersji źródeł jądra



Rysunek 3: Szczegółowe menu wyboru wersji źródeł jądra

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).



Rysunek 4: Wyszukiwanie nazwy

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.



Rysunek 5: Wyniki wyszukiwania nazwy

Po kliknięciu w pierwszy odnośnik w sekcji *Defined in 3 files*: użytkownik zostanie przeniesiony do strony z deklaracją funkcji. Jej prototyp (nagłówek) został oznaczony na żółto na rysunku 6.

```

Filter tags / include/linux/sched Search Identifier
v4
v4.16
v4.15
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
v4.6
v4.5
v4.4
v4.3
v4.2
v4.1
v4.0
v3
v2.6
172 extern long schedule_timeout(long timeout);
173 extern long schedule_timeout_interruptible(long timeout);
174 extern long schedule_timeout_killable(long timeout);
175 extern long schedule_timeout_uninterruptible(long timeout);
176 extern long schedule_timeout_idle(long timeout);
177 extern long schedule_timeout_lidle(long timeout);
178 asmlinkage void schedule(void);
179 extern void schedule_preempt_disabled(void);
180
181 extern int __must_check to_schedule_prepare(void);
182 extern void to_schedule_finish(int token);
183 extern long to_schedule_timeout(long timeout);
184 extern void to_schedule(void);
185
186 /**
187  * struct prev_cputime - snapshot of system and user cputime
188  * @utime: time spent in user mode
189  * @stime: time spent in system mode
190  * @lock: protects the above two fields
191  *
192  * Stores previous user/system time values such that we can guarantee
193  * monotonicity.
194  */
195 struct prev_cputime {
196 #ifdef CONFIG_VIRT_CPU_ACCOUNTING_NATIVE
197     u64 utime;
198     u64 stime;
199     raw_spinlock_t lock;
200 #endif
201 };
202
203 /**
204  * struct task_cputime - collected CPU time counts
205  * @utime: time spent in user mode, in nanoseconds
206  * @stime: time spent in kernel mode, in nanoseconds
207  * @blk_exec_runtime: total time spent on the CPU, in nanoseconds
208  *
209  * This structure groups together three kinds of CPU time that are tracked for
210  * threads and thread groups. Most things considering CPU time want to group
211  * these counts together and treat all three of them in parallel.
212  */
213 struct task_cputime {
214     u64 utime;
215     u64 stime;

```

Rysunek 6: Deklaracja funkcji schedule()

Po kliknięciu w drugi odnośnik we wspomnianej sekcji użytkownik zostanie przeniesiony do strony z definicją tej funkcji, która została zaznaczona na żółto na rysunku 7.

```

Filter tags / kernel/sched/core.c Search Identifier
v4
v4.16
v4.15
v4.14
v4.13
v4.12
v4.11
v4.10
v4.9
v4.8
v4.7
v4.6
v4.4.117
v4.4.116
v4.4.115
v4.4.114
v4.4.113
v4.4.112
v4.4.111
v4.4.110
v4.4.109
v4.4.108
v4.4.107
v4.4.106
v4.4.105
v4.4.104
v4.4.103
v4.4.102
v4.4.101
v4.4.100
v4.4.99
v4.4.98
3196  * make sure to submit it to avoid deadlocks.
3197  */
3198  if (blk_needs_flush_plug(task))
3199      blk_schedule_flush_plug(task);
3200 }
3201
3202 asmlinkage __visible void __sched schedule(void)
3203 {
3204     struct task_struct *task = current;
3205     sched_submit_work(task);
3206     do {
3207         preempt_disable();
3208         __schedule(false);
3209         sched_preempt_enable_no_resched();
3210     } while (need_resched());
3211     EXPORT_SYMBOL(schedule);
3212
3213 #ifdef CONFIG_CONTEXT_TRACKING
3214     asmlinkage __visible void __sched schedule_user(void)
3215     {
3216         /*
3217          * If we come here after a random call to set_need_resched(),
3218          * or we have been woken up remotely but the IPI has not yet arrived,
3219          * we haven't yet exited the NMI life mode. Do it here manually until
3220          * we find a better solution.
3221          *
3222          * NB: There are happy callers of this function. Ideally we
3223          * should warn if prev_state != CONTEXT_USER, but that will trigger
3224          * too frequently to make sense yet.
3225          */
3226         enum ctx_state prev_state = exception_enter();
3227         schedule();
3228         exception_exit(prev_state);
3229     }
3230 #endif
3231 }
3232
3233 /**
3234  * schedule_preempt_disabled - called with preemption disabled
3235  *

```

Rysunek 7: Definicja funkcji schedule()