# Systemy operacyjne

Lista zadań nr 2

Na zajęcia 17 października 2023

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Tanenbaum (wydanie czwarte): 10.3, 11.4
- Stallings (wydanie dziewiąte): 4.6
- APUE (wydanie trzecie): 8, 10

**UWAGA!** Każdy student na potrzeby prezentacji zadania ma uruchomić program emulatora terminala w trybie pełnoekranowym. Należy wybrać dużą czcionkę, kontrastowe kolory i jasne tło. Bądź uprzejmy wobec osób, które są oddalone od ekranu!

**Zadanie 1.** Na podstawie rysunku 4.15 z §4.6 przedstaw **stany procesu** w systemie Linux. Podaj akcje albo zdarzenia wyzwalające zmianę stanu. Które przejścia mogą być rezultatem działań podejmowanych przez: jądro systemu operacyjnego, kod sterowników, proces użytkownika? Wyjaśnij różnice między **snem przerywalnym** i **nieprzerywalnym**. Czy proces może **zablokować** lub **zignorować** sygnał «SIGKILL» lub «SIGSEGV»?

**Zadanie 2.** Wyjaśnij różnice w tworzeniu procesów w systemie Linux (§10.3.3) i WinNT (§11.4.3). Naszkicuj przebieg najważniejszych akcji podejmowanych przez jądro w trakcie obsługi funkcji `fork(2)` i `execve(2)`. Załóżmy, że system posiada wywołanie `spawn`, o takich samych argumentach jak `execve`. Zastępuje ono parę wywołań `fork` i `execve`, a realizuje takie samo zadanie. Dlaczego w takim przypadku mielibyśmy problemy z dodaniem do powłoki obsługi **przekierowania** standardowego wejścia/wyjścia odpowiednio z/do pliku albo łączenia dowolnych procesów **potokami**?

**Zadanie 3.** Na podstawie dokumentacji `fork(2)` (§8.3) i `execve(2)` (§8.10) wymień najważniejsze zasoby procesu, które są (a) dziedziczone przez proces potomny (b) przekazywane do nowego programu załadowanego do przestrzeni adresowej. Czemu przed wywołaniem `fork` należy opróżnić bufory biblioteki `stdio(3)`? Co jądro robi w trakcie wywołania `execve` z konfiguracją **zainstalowanych** procedur obsługi sygnałów?

**Zadanie 4.** Uruchom program «xeyes» po czym użyj na nim polecenia «kill», «pkill» i «xkill». Który sygnał jest wysyłany domyślnie? Przy pomocy kombinacji klawiszy «CTRL+Z» wyślij «xeyes» sygnał «SIGTSTP», a następnie wznów jego wykonanie. Przeprowadź inspekcję pliku «/proc/pid/status» i wyświetl maskę **sygnałów oczekujących** na dostarczenie. Pokaż jak będzie się zmieniać, gdy będziemy wysyłać wstrzymanemu procesowi kolejno: «SIGUSR1», «SIGUSR2», «SIGHUP» i «SIGINT». Co opisują pozostałe pola pliku «status» dotyczące sygnałów? Który sygnał zostanie dostarczony jako pierwszy po wybudzeniu procesu?

**Zadanie 5.** Na podstawie kodu źródłowy sinit.c[1] opowiedz jakie zadania pełni minimalny program rozruchowy `sinit`. Jakie akcje wykonuje pod wpływem wysyłania do niego sygnałów wymienionych w tablicy «sigmap»? Do czego służą procedury `sigprocmask(2)` i `sigwait(3)`? W jaki sposób grzebie swoje dzieci?

[1] https://git.suckless.org/sinit/files.html

Ściągnij ze strony przedmiotu archiwum «so21_lista_2.tar.gz», następnie rozpakuj i zapoznaj się z dostarczonymi plikami.

**UWAGA!** Można modyfikować tylko te fragmenty programów, które zostały oznaczone w komentarzu napisem «TODO».

**Zadanie 6.** Uzupełnij program «reaper.c» prezentujący powstawanie **sierot**. Proces główny przyjmuje rolę **żniwiarza** (ang. *reaper*) przy użyciu prctl(2). Przy pomocy procedury «spawn» utwórz kolejno procesy syna i wnuka. Następnie osieroć wnuka kończąc działanie syna. Uruchom podproces wywołujący polecenie «ps», aby wskazać kto przygarnął sierotę – przykład poniżej (zwróć uwagę na numery grup procesów):

```
  PID  PPID  PGRP STAT CMD
24886 24643 24886 S+   ./reaper (main)
24888 24886 24887 S    ./reaper (grandchild)
24889 24886 24886 R+   /usr/bin/ps -o pid,ppid,pgrp,stat,cmd
```

Po udanym eksperymencie należy zabić wnuka sygnałem «SIGINT», a następnie po nim posprzątać drukując jego **kod wyjścia**. Wysłanie «SIGINT» do procesu głównego jest zabronione! Zauważ, że proces główny nie zna numeru pid wnuka. W rozwiązaniu należy wykorzystać setpgid(2), pause(2), waitpid(2) i kill(2).

**UWAGA!** Użycie funkcji sleep(3) lub podobnych do właściwego uszeregowania procesów jest zabronione!

**Zadanie 7.** Uzupełnij program «cycle.c», w którym procesy grają w piłkę przy pomocy sygnału «SIGUSR1». Proces główny tworzy n dzieci. Każde z nich czeka na piłkę, a po jej odebraniu podaje ją do swojego starszego brata. Zauważ, że najstarszy brat nie zna swojego najmłodszego rodzeństwa, ale zna je ojciec – więc należy go wciągnąć do gry! Niech tata rozpocznie grę rzucając piłkę do najmłodszego dziecka. Kiedy znudzi Ci się obserwowanie procesów grających w piłkę możesz nacisnąć «CTRL+C» co wyśle «SIGINT» do całej rodziny. Możesz wprowadź do zabawy dodatkową piłkę wysyłając sygnał «SIGUSR1» poleceniem «kill». Czy piłki ostatecznie skleją się w jedną? W rozwiązaniu należy wykorzystać sigprocmask(2), sigsuspend(2) i kill(2).

**UWAGA!** Użycie funkcji sleep(3) lub podobnych do właściwego uszeregowania procesów jest zabronione!

**Zadanie 8.** Uzupełnij program «demand» o **procedurę obsługi sygnału** «SIGSEGV». Program ma za zadanie demonstrować przechwytywanie **błędów stron**, których nie było w stanie obsłużyć jądro SO.

Obsługujemy zakres adresów od «ADDR_START» do «ADDR_END». Pod losowo wybrane **wirtualne strony** z podanego przedziału zostanie podpięta **pamięć wirtualna** w trybie tylko do odczytu. Następnie program wygeneruje do zadanego przedziału adresów zapisy, które zakończą się naruszeniem ochrony pamięci.

Po wyłapaniu sygnału «SIGSEGV», korzystając z procedur «mmap_page» i «mprotect_page» odpowiednio zmapuj brakującą stronę (błąd «SEGV_MAPERR») i odblokuj zapis do strony (błąd «SEGV_ACCERR»). Dostęp do adresów spoza ustalonego zakresu powinien skutkować zakończeniem programu. Należy wtedy ustalić właściwy kod wyjścia tak, jakby proces został zabity sygnałem!

```
...
Fault at rip=0x55cb50d54389 accessing 0x10003fc0! Make page at 0x10003000 writable.
Fault at rip=0x55cb50d54389 accessing 0x10007bb0! Map missing page at 0x10007000.
...
Fault at rip=0x55cb50d5439c accessing 0x10010000! Address not mapped - terminating!
```

W procedurze obsługi sygnału można używać tylko procedur **wielobieżnych** (ang. *reentrant*) – sprawdź w podręczniku ich listę. Możesz wykorzystać procedurę «safe_printf», będącą okrojoną wersją «printf». Czemu można ją bezpiecznie wywołać w wnętrza «sigsegv_handler»?

Adres powodujący błąd strony i rodzaj błędu znajdziesz w argumencie «sigsegv_handler» o typie «siginfo_t», który opisano w podręczniku sigaction(2). Wskaźnik instrukcji, która spowodowała błąd strony, można przeczytać ze struktury przechowującej kontekst procesora «uc->uc_mcontext». Odpowiednie definicje znajdują się w pliku nagłówkowym «/usr/include/x86_64-linux-gnu/sys/ucontext.h».