Systemy komputerowe

Lista zadań nr B

Na zajęcia 1 – 4 czerwca 2020

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Tanenbaum (wydanie czwarte): 1.5, 2.1, 10.3,
- Arpaci-Dusseau: Introduction[1] Processes[2], Process API[3], Address Spaces[4]

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytłuszczoną** czcionką.
Podręcznikiem do zadań praktycznych/prezentacyjnych jest „Advanced Programming in the UNIX Environment" (w skrócie APUE). Proszę przede wszystkim korzystać z podręcznika systemowego (polecenia `man` i `apropos`). Jeśli jego treść w *Linuksie* nie jest wystarczająco klarowna, to warto spojrzeć do odpowiednika z BSD – najlepiej z użyciem strony mdoc.su.
W zadaniach oznaczonych literą **(P)**, wymagamy praktycznej **demonstracji** działania programów, najlepiej poprzez udostępnienie ekranu własnego komputera w usłudze Google Meet. By taka prezentacja była płynna należy przygotować w domu plik tekstowy z listą poleceń do wykonania i komentarzami. Każde zadanie należy mieć właściwie przygotowane do prezentacji przed zajęciami. W przypadku zbędnego przeciągania czasu odpowiedzi ze względu na problemy techniczne prowadzący ma prawo skreślić zadanie i postawić jeden punkt ujemny.

Zadanie 1. Podaj główne motywacje projektantów systemów operacyjnych do wprowadzenia **procesów** i **wątków**? Wymień główne różnice między nimi – rozważ współdzielone **zasoby**.

Zadanie 2. Wymień mechanizmy sprzętowe niezbędne do implementacji **wywłaszczania** (ang. *preemption*). Jak działa **algorytm rotacyjny** (ang. *round-robin*)? Jakie zadania pełni **planista** (ang. *scheduler*) i **dyspozytor** (ang. *dispatcher*)? Który z nich realizuje **politykę**, a który **mechanizm**?

Zadanie 3 (P). Do czego służy system plików proc[5] w systemie *Linux*? Znajdź identyfikator `PID` jednego ze swoich procesów, po czym wydrukuj zawartość katalogu `/proc/PID`. Wyświetl plik zawierający **argumenty wywołania**, **zmienne środowiskowe** i opis przestrzeni adresowej badanego procesu (`maps`). Wyjaśnij znaczenie kolumn (`man proc`) i wskaż, gdzie znajduje się sterta, stos, segment `text`, `data` oraz `bss` oraz procedury dynamicznego konsolidatora.

Zadanie 4. Na podstawie slajdu 25[6] przedstaw **stany procesu** w systemie *Linux*. Jakie akcje lub zdarzenia **synchroniczne** i **asynchronicznych** wyzwalają zmianę stanów? Kiedy proces opuszcza stan **zombie**? Wyjaśnij, które przejścia mogą być rezultatem działań podejmowanych przez: jądro systemu operacyjnego, kod sterowników, proces użytkownika albo administratora.

Zadanie 5. Jaką rolę pełnią **sygnały** w systemach uniksowych? W jakich sytuacjach jądro wysyła sygnał procesowi? Kiedy jądro **dostarcza** sygnały do procesu? Co musi zrobić proces by **wysłać sygnał** albo **obsłużyć sygnał**? Których sygnałów nie można **zignorować** i dlaczego? Podaj przykład, w którym obsłużenie sygnału `SIGSEGV` lub `SIGILL` może być świadomym zabiegiem programisty.

[1] `http://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf`
[2] `http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-intro.pdf`
[3] `http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-api.pdf`
[4] `http://pages.cs.wisc.edu/~remzi/OSTEP/vm-intro.pdf`
[5] `http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html`
[6] `https://skos.ii.uni.wroc.pl/pluginfile.php/32298/mod_resource/content/1/sk20-17-procesy-2.pdf`

Zadanie 6 (P). 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 «SIGSTOP», a następnie wznów jego wykonanie. Przeprowadź inspekcję pliku «/proc/$PID[7]/status» i wyświetl **maskę sygnałów** zgłoszonych procesowi (ang. *pending signals*). Pokaż jak będzie się zmieniać, gdy będziemy wysyłać wstrzymanemu procesowi po kolei: «SIGUSR1», «SIGUSR2», «SIGHUP» i «SIGINT». Co opisują pozostałe pola pliku «status» dotyczące sygnałów? Który sygnał zostanie dostarczony jako pierwszy po wybudzeniu procesu?

Zadanie 7 (P). Wbudowanym poleceniem powłoki «time» zmierz **czas wykonania** długo działającego procesu, np. polecenia «find /usr». Czemu suma czasów user i sys (a) nie jest równa real (b) może być większa od real? Poleceniem «ulimit» nałóż **ograniczenie** na **czas wykonania** procesów potomnych powłoki tak, by limit się wyczerpał. Uruchom ponownie wybrany program – który sygnał wysłano do procesu?

Zadanie 8 (P). Zaprezentuj sytuację, w której proces zostanie **osierocony**. W terminalu uruchom dodatkową kopię powłoki `bash`. Z jej poziomu wystartuj w tle[8] program `xclock` i sprawdź, kto jest jego rodzicem. Poleceniem `kill` wyślij sygnał `SIGKILL` do uruchomionej wcześniej powłoki i sprawdź, kto stał się nowym rodzicem procesu `xclock`. Wyjaśnij przebieg i wyniki powyższego eksperymentu. Co się stanie, gdy zamiast `SIGKILL` wyślemy powłoce sygnał `SIGHUP`?

Zadanie 9 (P). Wyświetl wykaz uruchomionych procesów wraz z ich **priorytetem** oraz wartością `nice`. Poleceniem `renice` spróbujemy wpłynąć na decyzje planisty zadań. Celem wygenerowania sztucznego obciążenia procesora uruchom polecenie `echo "" | awk '{for(;;) {}}'` w kilku terminalach jednocześnie. Wystartuj przeglądarkę `firefox` i z użyciem `renice` zmniejsz jej priorytet. Jak zmienił się **czas reakcji procesu**? Czy możesz przywrócić wartość `nice` do poprzedniego stanu?

[7] Zastąp $PID identyfikatorem uruchomionego procesu!
[8] Dodając znak & na końcu linii poleceń.