Systemy komputerowe

Lista zadań B

Karolina Szlęk 300411

Zadanie 1

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

Proces – proces to po prostu uruchomiony program, czyli egzemplarz wykonywanego programu. Po powstaniu, każdy proces dostaje unikatowy numer, który go identyfikuje, ten numer nazywany jest PID.
Idea stojąca za procesami polegała na tym, że w razie awarii jednego programu nie zakłóci ona działania innych, a po zakończeniu pracy, programu, jego zasoby wrócą do środowiska. Zaletą przydzielania zasobów jest to, że nie musi on (program) kontrolować jaka jego część znajduje się w pamięci fizycznej. Współdzielenie urządzenie z innymi programami też nie ma wtedy znaczenia. Procesy charakteryzuje też to, że zamiast bloków dyskowych udostępniają pliki.

Wątek –jest częścią programu wykonywaną współbieżnie w obrębie jednego procesu. Wątki z danego proces współdzielą przestrzeń adresową i innych struktur systemowych. Jeden proces może mieć wiele wątków. W porównaniu z procesami, wątki potrzebują mniejszą ilość zasobów do działania oraz mają mniejszy czas ich tworzenia. Ponadto używanie wątków jest początkowo łatwiejsze. Wątki jednego zadania mogą się między sobą łatwo komunikować (jest tak dzięki współdzieleniu przestrzeni adresowej). Nie ma potrzeby martwić się przekazywaniem dużej ilości danych, bo wystarczy przesłać wskaźnik.

Zasoby – jest to każdy jeden element systemu, który potencjalnie może być niezbędny do realizacji przetwarzania np.: pamiec, czas procesora, dostęp do urządzeń wejścia/wyjścia i pliki.

Różnica między procesami a wątkami
Procesy posiadają niezależne zasoby. Natomiast wątki, które działają w danym procesie współdzielą zasoby. Wątek jest częścią skład działającego procesu i używa zasobów nadanych procesowi. Proces jest działający program, któremu system przeznaczył zasoby. Tworzenie wątku jest znacząco szybsze niż procesu, a jego zakończenie nie zwalnia pamięci. W przypadku procesu pamięć jest zwalniana.

Zadanie 2

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**?

Wywłaszczenie (ang. preemption) polega na chwilowym przerwaniu procesu wykonywanego przez system komputerowy, przy zamiarze wznowienia procesu w późniejszym terminie. Wywłaszczenie stosowane jest w środowiskach wielozadaniowych, w której algorytm szeregujący może wstrzymać aktualnie wykonywane zadanie (np. proces lub wątek), po to, żeby inne zadanie mogło działać. Nie powoduje to blokady całego systemu, podczas zawieszenia jednego procesu. Do implementacji wywłaszczenia potrzebujemy mechanizmu okresowego przerwania (do przerwanie wykonywania procesu i przekazanie kontroli do systemu). Wykonanie cykliczne.

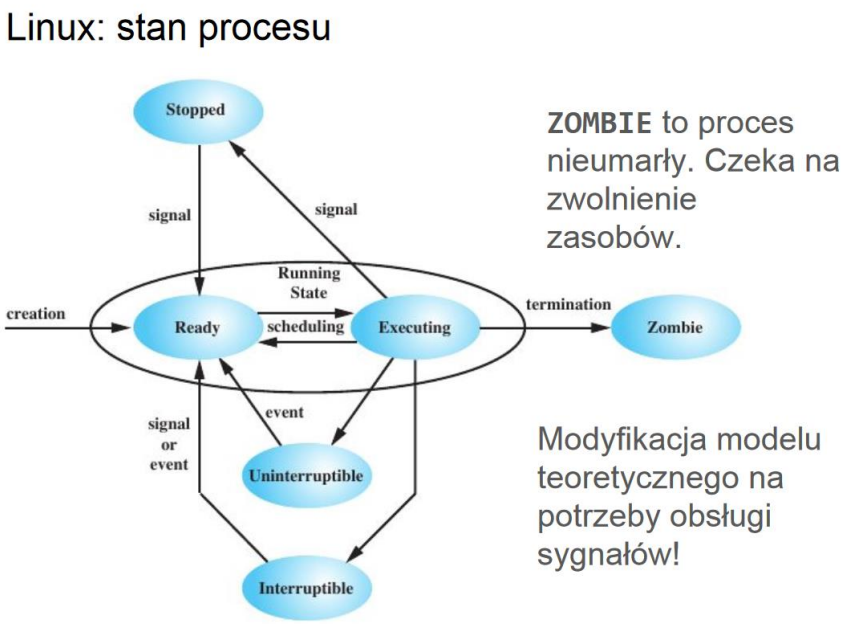
Algorytm rotacyjny (ang. round-robin) dla każdego procesu przydziela taki sam fragment czasu. Gdy proces zakończy działanie lub nastąpi wyjątek, albo po prostu czas upłynie, to następny proces korzysta z procesora itd. Algorytm ten jest wykorzystywany do udostępniania zasobu, tj zarządzania czasem korzystania z procesora.
Sposób działania: Planista kieruje pierwszy proces z kolejki (FIFO) do procesora i ustawiα czasomierz. Po przerwaniu od czasomierza uaktywnia się planisty, który odsyła przerwany proces na koniec kolejki. Do procesora znowu kieruje proces z początku kolejki. Sprawia to, że czas oczekiwania w kolejce jest proporcjonalny do liczby oczekujących.

Planista (ang. Scheduler)- część systemu operacyjnego. Odpowiada za wybieranie procesu, który w danym momencie ma mieć przydzielony dostęp do procesora. Może rozpocząć nowy proces, wstrzymać uruchomiony proces, przenieść proces na tył kolejki (realizuje polityke).

Dyspozytor (ang. Dispatcher) - procesowi wybranemu przez harmonogram daje kontrolę nad CPU. Jest wywoływany zawsze, gdy następuje context-switch. Odpowiada za zapisanie stanu zatrzymywanego procesu i wznowienie stanu procesu wskazanego przez Planiste.
Planista wybiera proces, a następnie dyspozytor ma oddać mu kontrolę nad procesorem. Po zatrzymaniu danego procesu odpowiada, za zapisanie jego aktualnego stanu i przywrócenia go po następnym wywołaniu (realizuje mechanizm).

Zadanie 4

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.



Stany procesu w systemie Linux., w oparciu o obrazek:

READY - proces gotowy do uruchomienia
EXECUTING - proces w trakcie jego wykonywania
SLEEPING - czeka na jakieś zdarzenie. Są dwa rodzaje:

- ***INTERRUPTIBLE*** (możliwy do przerwania)
  UNINTERRUPTIBLE (przerwanie snu niemożliwe)

STOPPED - proces zatrzymany przez sygnału SIGSTOP. W tym czasie może obdłużyć jedynie sygnały SIGKILL (zabicie procesu) i SIGCONT (przywrócenie procesu). Np. po wciśnięciu CTRL +Z
ZOMBIE proces po zakończeniu, ale jeszcze nie usunięty ze względu na przechowywane zasoby. Usunięcie procesu doprowadzi do zwolnienie pamięci i koniec koców utratę tych informacji. Gdy proces nadrzędny naszego procesu odczyta wszystkie potrzebne mu informacje, to proces wychodzi ze stanu zombie.

Zdarzenia synchroniczne, jest to takie zdarzenie, gdy następuje przejście ze stanu wykonywany w stan uśpiony.
Zdarzenia asynchroniczne, to otrzymanie sygnału np. SIGSTOP

Kiedy proces opuszcza stan *zombie*?

Proces opuszcza stan zombie, w momencie, gdy wszystkie potrzebne zasoby są juz zapisane, a sam proces usunięty z kolejki.

Jakie akcje lub zdarzenia ***synchroniczne*** i ***asynchronicznych*** wyzwalają zmianę stanów?

EXECUTING -> STOPPED
Asynchroniczne, inicjatorem działania jest administrator, np.CTRL + Z

STOPPED -> READY
Asynchroniczne, inicjatorem działania jest proces, np. SIGCONT

SLEEPING -> READY
Synchroniczne, inicjatorem działania jest kod sterownika, np. Proces dostaje zasoby, na które czekał

READY -> EXECUTING
Synchroniczne, inicjatorem działania jest jądro systemu, np. Wstrzymanie poprzedniego procesu, przydzielenie czasu nowemu

EXECUTING -> SLEEPING
Synchroniczne, inicjatorem działania jest jądro systemu, np. oczekiwanie na dane I/O

EXECUTING -> ZOMBIE
Synchroniczne, inicjatorem działania jest jądro systemu, np. Zatrzymanie procesu, ale jego dane trzeba będzie pobrać jeszcze

Zadanie 5

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.

Sygnał jest generowany przez jądro systemu, bądź też przez inny proces. Potrzebny jest do przekazania procesowi informacji o tym co się wydarzyło w systemie. Jest asynchroniczny. Niektóre sygnały można zignorować, ale są też takie, jak SIGSTOP czy SIGKILL. Informują nas one odpowiednio o zatrzymaniu, zabiciu procesu i nie można ich zignorować. To, że są one nie do zatrzymania wydaje się logiczne, ponieważ w przeciwnym razie doprowadzilibyśmy do sytuacji, w której programu nie da się zatrzymać.
Sygnały są używane do zadań takich jak kończenie działania procesów, czyinformowanie demonów, że mają odczytać ponownie pliki konfiguracyjne.
Najbardziej znanym przykładem sytuacji, w której jądro wysyła sygnał procesowi będzie naciśnięcie CTRL+Z, co spowoduje przerwanie procesu, ale sygnał będzie wysłany też, gdy na przykład będzie próba podzielenia przez 0 (niedozwolona operacja).
Jądro sprawdza czy należy doręczyć sygnał w kilku przypadkach. Przede wszystkim sprawdza czy należy doręczyć sygnał przed wejściem i po wyjściu w/ze stan uśpienia.
Proces może wysyłać sygnały w następujący sposób - do wybranego procesu -> wywołaniem kill.
Proces może też wysłać wysyłać opóźniony sygnały do samego siebie, wywołaniem alarm. Bądź też używając ualarm – proces będzie przesyłał sygnał sam do siebie, w równych interwałach.
Istnieje też opcja umożliwienia procesowi obsługi sygnałów. Wystarczy zdefiniować procedurę obsługi sygnałów, a następnie przekazać o niej informacje do systemu. Można to zrobić wywołaniem void(*signal(int sig, void(*func)(int)))(int)

Podaj przykład, w którym obsłużenie sygnału ***SIGSEGV*** lub ***SIGILL*** może być świadomym zabiegiem programisty.

Wszelkie niechcianego zachowania programu - chcemy móc wyłapać informację o ich wystąpieniu. Dlatego właśnie potrzebujemy obsługiwać następujące sygnały: SIGSEGV lub SIGILL. Pozwoli nam to w przyszłości móc naprawić program. np

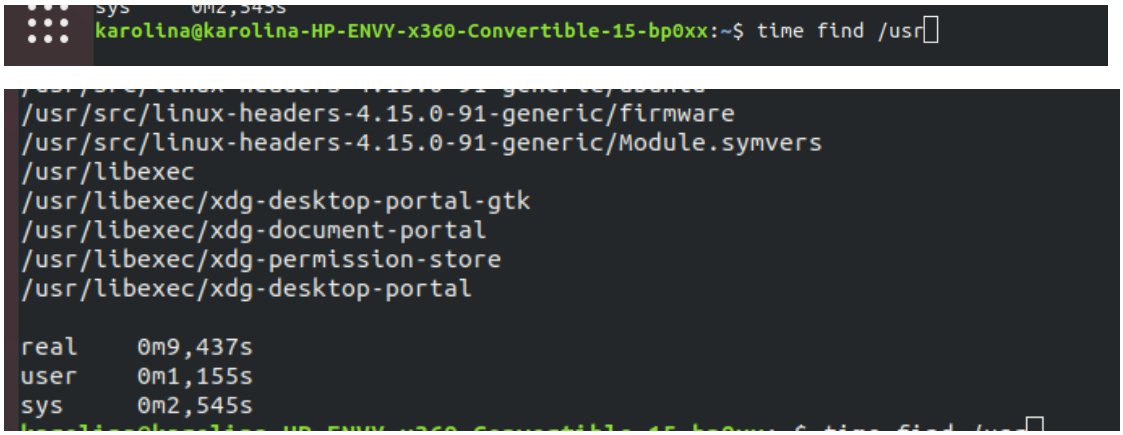
Serwery www działają cały czas. Chcemy uruchomić nowy proces i zabić bieżący, żeby pomimo wystąpienia błędu, który powinien uniemożliwić działanie procesu, utrzymać działanie usługi.

Zadanie 7

> **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?

Pomocne definicje:

real - czas liczony od momentu włączenia do chwili uzyskania wyniku, jest czasem rzeczywistym, ze wszystkimi oczekiwaniami itp..

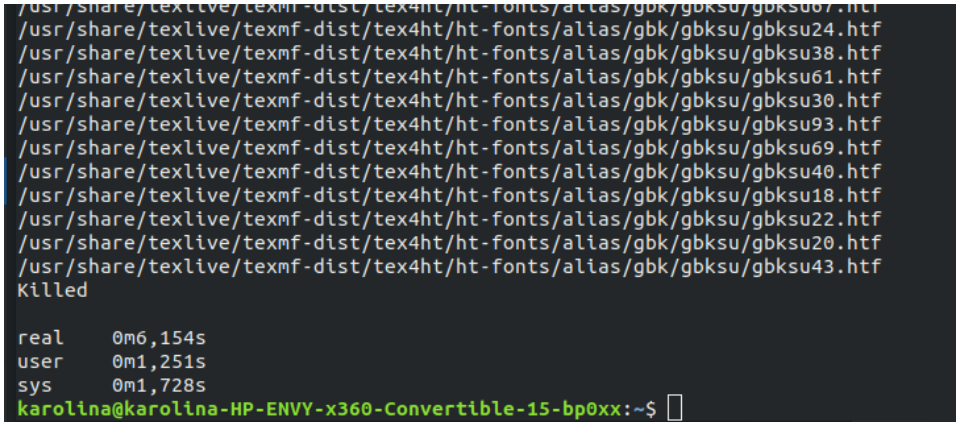


Czemu suma czasów user i sys nie jest równa real?
Suma czasów user i sys będzie różna niż real, ponieważ, w real musi zostać jeszcze uwzględniony czas oczekiwania procesu na swoją kolej wykonywania lub odczyt z pamięci.

Czemu suma czasów user i sys może być większa od real?
Jeżeli pracujemy na kilku procesach, to user i sys zsumują czas wszystkich procesów. Real tego nie zrobi, przez co będzie mniejszy niz user i sys.

ulimit -t 1

time find /usr



```
karolina@karolina-HP-ENVY-x360-Convertible-15-bp0xx:~$ echo $?
```

```
sys     0m0,626s
karolina@karolina-HP-ENVY-x360-Convertible-15-bp0xx:~$ echo $?
137
karolina@karolina-HP-ENVY-x360-Convertible-15-bp0xx:~$
```

echo $? to kod powrotu z procesu ostatniego uruchomienia. 0 oznacza, że nie wystąpił błąd. Inne wartości reprezentują jakiś niezwykły stan

```
karolina@karolina-HP-ENVY-x360-Convertible-15-bp0xx:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Naszym jest:

9) SIGKILL - Zakończenie procesu
Jest to jedyny całkowicie pewny sposób zakończenia wykonywania procesu, ponieważ proces odbierający ten sygnał nie może go ani zignorować, ani przechwycić (dostarczając własną funkcję obsługi sygnału). Sygnału tego powinno sie używać tylko w sytuacjach wyjątkowych, w pozostałych natomiast zalecany jest sygnał SIGTERM.