Systemy operacyjne

Mariusz Żynel
mariusz@math.uwb.edu.pl
http://math.uwb.edu.pl/~mariusz/

Uniwersytet w Białymstoku

2023/2024

Proces

- Egzemplarz wykonywanego programu
- Jeden program (kod programu), wiele procesów (np. httpd)
- Zasoby (pamięć, czas procesora, urządzenia wejścia-wyjścia, pliki)
- Zasoby mogą być współdzielone, poza czasem procesora
- Przestrzeń adresowa
- Zarządzanie procesami odbywa się w jądrze systemu operacyjnego
- Wszystko w Unixie jest plikiem, także proces
- Uprawnienia związane z procesami są analogiczne jak z plikami

Struktura opisująca proces

`PID`	identyfikator procesu
`PPID`	identyfikator procesu rodzica
`UID, EUID`	rzeczywiste i efektywne ID użytkownika
`GID, EGID`	rzeczywiste i efektywne ID grupy
`PRI`	priorytet
`NI`	wartość NICE do obliczeń priorytetu
`ADDR`	adres pamięci procesu
`SZ/SIZE`	rozmiar procesu w pamięci wirtualnej
`WCHAN`	adres zdarzenia, dla którego proces jest uśpiony
`STIME`	czas uruchomienia
`TIME`	efektywny, sumaryczny czas procesora
`TTY`	terminal kontrolujący proces, znak ? oznacza brak terminala
`CMD`	nazwa programu, argumenty
`NLWP`	ilość wątków
`STATE`	stan procesu

Tworzenie procesu

- Utworzenie przestrzeni adresowej
- Wypełnienie struktury opisującej proces
- Kopiowanie kodu i danych z programu wykonywalnego do przestrzeni adresowej procesu
- Mapowanie współdzielonych zasobów w przestrzeń adresową procesu
- Dołączenie procesu do kolejki procesów oczekujących na przydzielenie czasu procesora i ustalenie priorytetu wykonania
- Ustawienie stanu procesu na działający

Stany procesu

- W trakcie tworzenia
- Aktualnie wykonywany przez procesor
- Czekający na dostęp do zasobów
- Uśpiony
- Przeznaczony do zniszczenia
- Zombie

Wątek (thread, lightweight process)

- Część programu wykonywana współbieżnie w ramach jednego procesu
- W jednym procesie może istnieć wiele wątków
- Współdzielenie przestrzeni adresowej i zasobów procesu
- Wątki wymagają mniej zasobów i krótszy jest czas ich tworzenia
- Wątki mogą komunikować się między sobą w łatwy sposób, bez pośrednictwa systemu operacyjnego
- Przekazanie dowolnie dużej ilości danych wymaga przesłania jedynie wskaźnika
- Równoczesny dostęp do wspólnych danych grozi utratą spójności danych i w konsekwencji błędem działania programu
- Synchronizacja wątków: semafory, muteksy, sekcje krytyczne

Dyspozytor (scheduler)

- Część jądra w systemach operacyjnych z podziałem czasu, odpowiedzialna za przydzielanie czasu procesora w ramach przełączania procesów
- Decyzja o tym, któremu procesowi przydzielić czas procesora jest podejmowana przez algorytm szeregowania (np. FIFO, round-robin)
- Jednym z zadań dyspozytora jest przełączanie kontekstu
- Głodzenie procesu – proces nie jest w stanie kontynuować działania, ponieważ nie ma dostępu do wymaganych zasobów
- Wywłaszczanie – wstrzymanie wykonywanego procesu, aby umożliwić działanie innemu procesowi, dzięki temu zawieszenie jednego procesu nie powoduje blokady całego systemu
- Przełączanie kontekstu – zachowywanie i odtwarzanie stanu procesora/rdzenia (kontekstu), aby wiele procesów mogło współdzielić zasoby

Informacje o procesach

`ps` raport o aktywnych procesach

- `-e` lista wszystkich procesów
- `-f` pełna informacja (UID, PPID, STIME)
- `-l` dodatkowe informacje (UID, PPID, PRI, NI, ADDR, SZ/SIZE, WCHAN)
- `-L` informacja o wątkach

`pgrep` odszukiwanie procesów według nazwy programu lub atrybutów

`top` statystyki aktywnych procesów

`prstat` statystyki aktywnych procesów, opcja -L pokazuje wątki (tylko Solaris)

`/proc` katalog z pełną informacją o wszystkich procesach

Komunikacja między procesami (IPC), sygnały

- Sygnał – powiadomienie wysłane z jednego procesu do drugiego
- Proces odbierający sygnał przerywa swoją czynność i natychmiast reaguje na odebrany sygnał w odpowiedni sposób
- Sygnały mają przypisane numery oraz nazwy symboliczne
- Listę wszystkich sygnałów możemy zobaczyć poleceniem `kill -l`

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGEMT       8) SIGFPE
 9) SIGKILL     10) SIGBUS      11) SIGSEGV     12) SIGSYS
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGUSR1
17) SIGUSR2     18) SIGCHLD     19) SIGPWR      20) SIGWINCH
21) SIGURG      22) SIGIO       23) SIGSTOP     24) SIGTSTP
25) SIGCONT     26) SIGTTIN     27) SIGTTOU     28) SIGVTALRM
29) SIGPROF     30) SIGXCPU     31) SIGXFSZ     32) SIGWAITING
33) SIGLWP      34) SIGFREEZE   35) SIGTHAW     36) SIGCANCEL
37) SIGLOST     41) SIGRTMIN    42) SIGRTMIN+1  43) SIGRTMIN+2
44) SIGRTMIN+3  45) SIGRTMAX-3  46) SIGRTMAX-2  47) SIGRTMAX-1
48) SIGRTMAX
```

Definicja sygnałów 1 – 19

Fragment pliku (w Solaris): `/usr/include/sys/iso/signal_iso.h`

```
#define SIGHUP  1       /* hangup */
#define SIGINT  2       /* interrupt (rubout) */
#define SIGQUIT 3       /* quit (ASCII FS) */
#define SIGILL  4       /* illegal instruction (not reset when caught) */
#define SIGTRAP 5       /* trace trap (not reset when caught) */
#define SIGIOT  6       /* IOT instruction */
#define SIGABRT 6       /* used by abort, replace SIGIOT in the future */
#define SIGEMT  7       /* EMT instruction */
#define SIGFPE  8       /* floating point exception */
#define SIGKILL 9       /* kill (cannot be caught or ignored) */
#define SIGBUS  10      /* bus error */
#define SIGSEGV 11      /* segmentation violation */
#define SIGSYS  12      /* bad argument to system call */
#define SIGPIPE 13      /* write on a pipe with no one to read it */
#define SIGALRM 14      /* alarm clock */
#define SIGTERM 15      /* software termination signal from kill */
#define SIGUSR1 16      /* user defined signal 1 */
#define SIGUSR2 17      /* user defined signal 2 */
#define SIGCLD  18      /* child status change */
#define SIGCHLD 18      /* child status change alias (POSIX) */
#define SIGPWR  19      /* power-fail restart */
```

Definicja sygnałów 20 – 40

```
#define SIGWINCH 20     /* window size change */
#define SIGURG  21      /* urgent socket condition */
#define SIGPOLL 22      /* pollable event occured */
#define SIGIO   SIGPOLL /* socket I/O possible (SIGPOLL alias) */
#define SIGSTOP 23      /* stop (cannot be caught or ignored) */
#define SIGTSTP 24      /* user stop requested from tty */
#define SIGCONT 25      /* stopped process has been continued */
#define SIGTTIN 26      /* background tty read attempted */
#define SIGTTOU 27      /* background tty write attempted */
#define SIGVTALRM 28    /* virtual timer expired */
#define SIGPROF 29      /* profiling timer expired */
#define SIGXCPU 30      /* exceeded cpu limit */
#define SIGXFSZ 31      /* exceeded file size limit */
#define SIGWAITING 32   /* reserved signal no longer used by threading code */
#define SIGLWP  33      /* reserved signal no longer used by threading code */
#define SIGFREEZE 34    /* special signal used by CPR */
#define SIGTHAW 35      /* special signal used by CPR */
#define SIGCANCEL 36    /* reserved signal for thread cancellation */
#define SIGLOST 37      /* resource lost (eg, record-lock lost) */
#define SIGXRES 38      /* resource control exceeded */
#define SIGJVM1 39      /* reserved signal for Java Virtual Machine */
#define SIGJVM2 40      /* reserved signal for Java Virtual Machine */
```

Wysyłanie sygnałów

`kill` wysyła sygnał do procesu wskazanego przez podanie PID

-SIGNAME wysyła sygnał `SIGNAME`, jeśli nie określono sygnału domyślnie wysyłane jest `SIGTERM` kończące proces

-l podaje pełną listę sygnałów, numer albo nazwę podanego sygnału

`pkill` wysyła sygnał do wszystkich procesów wskazanych przez podanie nazwy programu lub atrybutów, należy uważać aby nazwa jednoznacznie określała proces

Przechwytywanie, ignorowanie i obsługa sygnałów

- Dwa sygnały są wyjątkowe:
  - `SIGKILL 9` : bezwarunkowe zakończenie procesu (kill)
  - `SIGSTOP 23` : wstrzymanie wykonania procesu (stop)
- Nie mogą one być ani przechwycone, ani zignorowane
- W swoim programie każdy inny sygnał możemy:
  - zignorować, albo
  - skojarzyć z nim obsługującą funkcję (handler)
- Do przechwytywania sygnałów
  - w shellu mamy wbudowaną funkcję `trap`
  - w bibliotece C mamy następujące funkcje:

```
#include <signal.h>

void (*signal(int sig, void (*disp)(int)))(int);
void (*sigset(int sig, void (*disp)(int)))(int);
int sighold(int sig);
int sigrelse(int sig);
int sigignore(int sig);
int sigpause(int sig);
```