Środowisko procesu Unixowego

■ Proces unixowy jest wykonywalnym programem załadowanym do pamięci operacyjnej komputera, wraz z pewnym środowiskiem (zestawem zmiennych z wartościami).

■ Jądro Unixa utrzymuje tablicę wszystkich istniejących procesów wraz z zestawem informacji o nich, np. zestawem otwartych plików, maską sygnałów procesu, priorytetem, i innymi.

■ Procesy tworzone są zawsze przez klonowanie istniejącego procesu funkcją `fork`, zatem każdy proces posiada tzw. proces nadrzędny, albo rodzicielski (*parent process*), który go utworzył. Wyjątkiem jest proces numer 1, utworzony przez jądro Unixa, i wykonujący program init, który jest protoplastą wszystkich innych procesów w systemie.

■ Grupa procesów: wszystkie podprocesy (*child processes*) uruchomione przez jeden proces nadrzędny.

■ Proces charakteryzują: pid,pgrp,uid,euid,gid,egid

Własności procesu i podprocesu

■ wspólne:

- otwarte pliki (deskryptory)
- real i effective UID i GID, proces group ID (PGRP)
- terminal i sesja
- kartoteka bieżąca i root
- maska praw dostępu tworzonych plików (umask)
- maska sygnałów i handlery
- środowisko (zbiór zmiennych środowiskowych)
- otwarte wspólne obszary pamięci
- ograniczenia zasobów

■ różne:

- PID
- wartość zwracana z funkcji fork
- blokady plików nie są dziedziczone
- ustawiony alarm procesu nadrzędnego jest kasowany dla podprocesu
- zbiór wysłanych (ale jeszcze nieodebranych) sygnałów jest kasowany dla podprocesu

Stany procesów

Stan	Ang.	Znaczenie
wykonywalny	*runnable*	proces w kolejce do wykonywania
uśpiony	*sleeping*	proces czeka na dostęp do zasobu
wymieciony	*swapped-out*	proces usunięty z pamięci
nieusuwalny	*zombie*	proces nie może się zakończyć
zatrzymany	*stopped*	wykonywanie procesu wstrzymane sygnałem

proces wymieciony jest normalnie w jednym z pozostałych stanów, lecz został usunięty z pamięci i aż do chwili ponownego załadowania nic nie może się z nim stać

Kończenie pracy procesów

- Proces unixowy kończy się normalnie albo anormalnie (np. przez otrzymanie sygnału), zawsze zwracając kod zakończenia, tzw. *status*.

- Rodzic procesu normalnie powinien odczytać kod zakończenia potomka wywołując funkcję systemową `wait`. Dopóki ten kod nie zostanie przeczytany, proces nie może umrzeć do końca i pozostaje w stanie zwanym *zombie*.

- Istnieje mechanizm adopcji polegający na tym, że procesy, których rodzic zginął przed nimi, zostają adoptowane przez proces `init`. `init` wywołuje okresowo funkcję `wait` aby umożliwić poprawne zakończenie swoich potomków.

- Gdy proces nadrzędny żyje w chwili zakończenia pracy potomka, i nie wywołuje funkcji `wait`, to potomek pozostaje w stanie *zombie* na czas nieograniczony, co może być przyczyną wyczerpania jakichś zasobów systemu (np. zapełnienia tablicy procesów).

Sygnały

■ Sygnały są mechanizmem asynchronicznego powiadamiania procesów o wydarzeniach.

■ Domyślną reakcją procesu na otrzymanie sygnału jest natychmiastowe zakończenie pracy (niektóre sygnały mają inne reakcje domyślne).

■ Proces może zadeklarować własną procedurę obsługi sygnału, lub zadeklarować ignorowanie sygnału (pewnych sygnałów nie można przechwycić ani ignorować).

Mechanizmy generowania sygnałów

- naciskanie pewnych klawiszy na terminalu użytkownika (SIGINT, SIGQUIT)
- wyjątki sprzętowe: nielegalna instrukcja, nielegalne odwołanie do pamięci, dzielenie przez 0, itp. (SIGILL, SIGSEGV, SIGFPE)
- wywołanie komendy kill przez użytkownika (SIGTERM, i inne)
- funkcje `kill` i `raise`
- mechanizmy software-owe (SIGALRM)

Obsługa sygnałów

- sprawdzenie na początku programu, czy dyspozycją sygnału nie jest ignorowanie (program może być wywoływany w specjalnym środowisku, ignorującym sygnały)

- ignorowanie sygnału na wejściu do handlera

- zakończeniem pracy handlera może być:
  - koniec procesu
  - kontynuacja
  - kontynuacja z przekazaniem informacji
  - kontynuacja od określonego punktu

- ponowne zadeklarowanie handlera w przypadku kontynuacji

Automatyczne uruchamianie procesów: `crontab`

```
/var/spool/cron/crontabs/root:
0 2 * * 0,4 /etc/cron.d/logchecker
5 4 * * 6   /usr/lib/newsyslog
30 23 * * 0 /usr/local/sbin/rotate-logs-weekly
30 22 1 * * /usr/local/sbin/rotate-logs-monthly
15 3 * * * /usr/lib/fs/nfs/nfsfind
1 2 * * * [ -x /usr/sbin/rtc ] && /usr/sbin/rtc -c
                                           > /dev/null 2>&1

/var/spool/cron/crontabs/sys:
0,10,20,30,40,50 * * * 0-6 /usr/lib/sa/sa1
5 22 * * 1-5 /usr/lib/sa/sa2 -s 8 -e 22 -i 1800 -A

QNX:/var/spool/cron/crontabs/root:
0,10,20,30,40,50 * * * * rtc -S 360 net 1
```


Automatyczne uruchamianie procesów: `at`

```
sierra-63> echo 'echo otwarcie pracowni \
                      | mailx -s "szkola, 15:00" witold' \
                      | at 11 am november 8
warning: commands will be executed using /usr/bin/tcsh
job 973677600.a at Wed Nov  8 11:00:00 2000


sierra-64> at -l
973677600.a     Wed Nov  8 11:00:00 2000
```

Selektywna autoryzacja użytkowników do używania `crontab` i `at`:
/usr/lib/cron/{cron,at}.{allow,deny}

Rozwiązywanie problemów z procesami

■ program systemowy `ps` wyświetla listę procesów z jednolinijkowymi informacjami o każdym procesie, co pozwala np. wykrywać procesy zużywające dużo zasobów (czas CPU, pamięć fizyczna)

→ różnice w wywołaniu i działaniu `ps` na różnych systemach (AT&T/BSD)

■ program `top` pokazuje zestaw najaktywniejszych procesów

→ również działa w zależności od systemu

■ ustawianie ograniczeń na zasoby procesu:
`RLIMIT_CORE,RLIMIT_CPU,RLIMIT_DATA,RLIMIT_FSIZE,RLIMIT_NOFILE`

→ bardzo duże rozbieżności między systemami

■ sterowanie priorytetami procesów: wartość `nice` procesu, komendy `nice` (użytkownik) i `renice` (administrator):

```
nice -n 20 pid-dlugiego-zadania </dev/null &
```

→ administrator może „nakłaniać” użytkowników do stosowania wyższych wartości `nice` przez zagnieżdżanie aplikacji systemowych w skryptach ustawiających wartość `nice`

■ znajdowanie procesów, które korzystają z danego pliku, lub katalogu (np. partycji dyskowej):

```
fuser /cdrom
```

■ informacje o plikach otwartych przez dany proces:
`pfiles` *pid*

■ „zabijanie” procesów przez wysłanie sygnału:
`kill` *pid*

■ domyślny sygnał TERM=15 może być przechwycony przez proces, wtedy:
`kill -9` *pid*

■ zatrzymywanie procesu „do wyjaśnienia”:
`kill -STOP` *pid*
aby kontynuować:
`kill -CONT` *pid*

Startowanie systemu

1. hardware

2. bootstrap z dysku (lub sieci)

3. jądro Unixa na dysku:

   (a) sprawdza stan hardware'u
   (b) montuje /
   (c) uruchamia init

4. init bootuje system wg specyfikacji w pliku `/etc/inittab`, w szczególności:

   (a) montuje dodatkowe file-systemy (plik `/etc/fstab`), np.: `/usr`, `/home`, ...
   (b) odpala procesy usługowe (daemony)

Stany pracy i gaszenie systemu

- tryby pracy systemu Unix: single-user, multi-user, inne

- normalny proces gaszenia: parametry *grace, init*, i komunikat:
  `shutdown -g 60 -i 6 "restart systemu dla rekonfiguracji"`
  po zatrzymaniu może nastąpić:

  – zwykłe zatrzymanie systemu: `-i 0`
  – wyłączenie zasilania, jeśli możliwe: `-i 5`
  – ponowny restart: `-i 6`
  – restart do trybu single-user: `-i 1` lub `-i s`

- szybki restart bez zatrzymywania procesów, ale sync
  `reboot`

- ekspresowe zatrzymanie systemu, jak reboot ale bez restartu
  `halt`

- zatrzymanie jak halt plus wyłączenie zasilania
  `poweroff`

Poziomy pracy Systemu V

konfiguracje uruchamianych procesów, definiowane w `/etc/inittab`

poziom 0: system wyłączony

poziom 1,S: tryb single-user, systemy plików dyskowych zamontowane, uruchomiony jest tylko minimalny zestaw procesów

poziom 2: tryb multi-user, uruchomione interface-y sieciowe (w starszych Unixach ten poziom nie uruchamiał oprogramowania sieciowego)

poziom 3: rozszerzenie poziomu 2, uruchamiane są wszystkie procesy poziomu 2, plus dodatkowe, np. sieciowy serwer plików (NFS)

poziom 4: poziom wielodostępu alternatywny do 3, pozwala wprowadzić inny zestaw procesów

poziom 5: tryb gaszenia systemu aż do wyłączenia zasilania

poziom 6: tryb gaszenia systemu aż do pełnego zatrzymania i natychmiastowy restart

Unix — pliki startowe: `/etc/inittab`

- `/etc/inittab` (System V, Linux)

```
id:3:initdefault:
si:S:sysinit:/etc/rc.d/rc.S
su:S:wait:/etc/rcS
l0:0:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
l1:1:wait:/etc/rc1 >/dev/console 2>&1 </dev/console
l2:2:wait:/etc/rc2 >/dev/console 2>&1 </dev/console
l3:3:wait:/etc/rc3 >/dev/console 2>&1 </dev/console
l5:5:wait:/etc/rc5 >/dev/console 2>&1 </dev/console
l6:6:wait:/etc/rc6 >/dev/console 2>&1 </dev/console
rc:123456:wait:/etc/rc.d/rc.M
ca::ctrlaltdel:/sbin/shutdown -t3 -rf now
pf:0123456:powerfail:/sbin/shutdown -f +5 "THE POWER IS FAILING"
pg:0123456:powerokwait:/sbin/shutdown -c "THE POWER IS BACK"
ps:S:powerokwait:/sbin/init 5

# Dialup lines
d1:2345:respawn:/sbin/mgetty -n 1 /dev/ttyS0 C38400
#d2:2345:respawn:/sbin/agetty 19200 ttyS1
b0:2345:respawn:/sbin/mgetty -n 1 /dev/ttyE0 C38400

x:5:respawn:/usr/bin/X11/xdm -nodaemon
```

Włączanie użytkownika do systemu

1. `init` uruchamia proces `getty`, który inicjuje port i oczekuje na połączenie użytkownika.
2. Użytkownik dokonuje fizycznego połączenia z interface-m komputera.
3. `getty` wykrywa włączonego użytkownika, wyświetla komunikat login, wczytuje wprowadzoną nazwę użytkownika, i uruchamia program `login`.
4. `login` czyta i sprawdza hasło użytkownika, i wywołuje właściwy interpreter komend (shell) ze środowiskiem zainicjowanym znanymi sobie parametrami.
5. Interpreter komend znajduje właściwe sobie pliki startowe (systemowe i własne) i ostatecznie konfiguruje środowisko użytkownika, a następnie przechodzi do pracy interakcyjnej.
6. Po zakończeniu pracy interpretera komend `init` wykrywa brak programu obsługi portu i ponownie uruchamia `getty` (w przypadku połączeń sieciowych oczekiwanie na połączenia trwa cały czas).