Zarządzanie procesami w systemie Unix

Witold Paluszyński
Katedra Cybernetyki i Robotyki
Politechnika Wrocławska
http://www.kcir.pwr.edu.pl/~witold/

Proces uniksowy i jego środowisko

- Proces uniksowy jest wykonywalnym programem załadowanym do pamięci operacyjnej komputera. Każdy proces uniksowy wykonuje się we własnej wirtualnej przestrzeni adresowej.
- Proces jest normalnie tworzony z trzema otwartymi plikami: stdin, stdout i stderr przypisanymi do terminala użytkownika interakcyjnego.
- Jądro Uniksa utrzymuje tablicę wszystkich istniejących procesów wraz z zestawem informacji kontekstowych o nich, np. zestawem otwartych plików, terminalem sterującym, priorytetem, maską sygnałów, i innymi.
- Proces posiada **środowisko**, które jest zestawem dostępnych dla procesu zmiennych środowiskowych z wartościami (tekstowymi). Środowisko nazywa się zewnętrznym ponieważ jest początkowo tworzone dla procesu przez system, a potem jest również dostępne na zewnątrz procesu.
- Proces tworzony jest z zestawem argumentów wywołania programu, który jest wektorem stringów
- Proces charakteryzują: pid, ppid, pgrp, uid, euid, gid, egid

Tworzenie procesów

- Procesy tworzone są zawsze przez klonowanie istniejącego procesu, zatem każdy proces posiada tzw. proces nadrzędny, lub inaczej rodzicielski (*parent process*), który go utworzył. Wyjątkiem jest proces numer 1, utworzony przez jądro Uniksa, i wykonujący program `init`, który jest protoplastą wszystkich innych procesów w systemie.
- Proces potomny łączą ścisłe związki z jego procesem rodzicielskim. Potomek dziedziczy szereg atrybutów procesu nadrzędnego: należy do tego samego użytkownika i grupy użytkowników, ma początkowo ten sam katalog bieżący, otrzymuje środowisko, które jest kopią środowiska rodzica, ma tę samą sesję i terminal sterujący, należy do tej samej grupy procesów, dziedziczy maskę sygnałów i handlery, ograniczenia zasobów, itp.
- Poza dziedziczeniem atrybutów rodzica, potomek współdzieli z nim pewne zasoby, z których najważniejszymi są otwarte pliki. Objawy tego można łatwo zauważyć, gdy procesy uruchamiane przez interpreter poleceń mogą czytać z klawiatury i pisać na ekranie, a gdy jest kilka procesów uruchomionych w tle, to ich operacje wejścia/wyjścia mogą się mieszać. (Innymi współdzielonymi obiektami są otwarte wspólne obszary pamięci.)

Kończenie pracy procesów

- Proces uniksowy kończy się normalnie albo anormalnie (np. przez otrzymanie sygnału), zawsze zwracając kod zakończenia, tzw. *status*.
- Po śmierci procesu jego rodzic normalnie powinien odczytać status potomka wywołując funkcję systemową `wait`. Dopóki status 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 (sieroty), zostają adoptowane przez proces nr 1 (`init`). `init` wywołuje okresowo funkcję `wait` aby umożliwić poprawne zakończenie swoich potomków (zarówno naturalnych jak i adoptowanych).
- 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).

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 wstrzymane sygnałem

- **Wymiatanie** procesów ma związek z funkcjonowaniem pamięci wirtualnej i jest objawem posiadania niewystarczającej ilości pamięci przez system. (Podstawowym mechanizmem odzyskiwania pamięci fizycznej jest **stronicowanie**). Wymiatanie polega na wytypowaniu pojedynczego procesu i chwilowym usunięciu go z kolejki procesów wykonujących się w celu odzyskania przydzielonej mu pamięci.

  Proces wymieciony jest normalnie w jednym z pozostałych stanów, lecz został usunięty z pamięci i aż do chwili ponownego załadowania nie może zmienić stanu.
- System przenosi proces w stan uśpienia jeśli nie może mu czegoś dostarczyć, np. dostępu do pliku, danych, itp. W stanie uśpienia proces nie zużywa czasu procesora, i jest *budzony* i wznawiany w sposób przez siebie niezauważony.

Grupy procesów

- **Grupa procesów:** wszystkie podprocesy (*child processes*) utworzone przez jeden proces nadrzędny. Każdy proces w chwili utworzenia automatycznie należy do grupy procesów swojego rodzica.
- Pod pewnymi względami przynależność do grupy procesów jest podobna do przynależności do partii politycznych. Każdy proces może:
  - założyć nową grupę procesów (o numerze równym swojemu PID),
  - wstąpić do dowolnej innej grupy procesów,
  - włączyć dowolny ze swoich podprocesów do dowolnej grupy procesów (ale tylko dopóki podproces wykonuje kod rodzica).
- Grupy procesów mają głównie znaczenie przy wysyłaniu sygnałów.

Ograniczenia konsumpcji zasobów dla procesu

System operacyjny może i powinien ograniczać wielkość zasobów zużywanych przez procesy. Systemy uniksowe definiują szereg zasobów, które mogą być limitowane dla procesu. Proces może swoje ograniczenia ustawiać funkcją systemową `ulimit`. Istnieje również polecenie wbudowane shella o tej nazwie.

ulimit	nazwa	kod	opis zasobu
-c	coredumpsize	RLIMIT_CORE	plik core (zrzut obrazu pamięci) w kB
-d	datasize	RLIMIT_DATA	segment danych procesu w kB
-f	filesize	RLIMIT_FSIZE	wielkość tworzonego pliku w kB
-l	memorylocked	RLIMIT_MEMLOCK	obszar, który można zablokować w pamięci w kB
-m	memoryuse	RLIMIT_RSS	zbiór rezydentny w pamięci w kB
-n	descriptors	RLIMIT_NOFILE	liczba deskryptorów plików (otwartych plików)
-s	stacksize	RLIMIT_STACK	wielkość stosu w kB
-t	cputime	RLIMIT_CPU	wykorzystanie czasu CPU w sekundach
-u	maxproc	RLIMIT_NPROC	liczba procesów użytkownika
-v	vmemoryuse	RLIMIT_VMEM	pamięć wirtualna dostępna dla procesu w kB

Dla każdego zasobu istnieją dwa ograniczenia: miękie i twarde. Obowiązujące jest ograniczenie miękie, i użytkownik może je dowolnie ustawiać, lecz tylko do maksymalnej wartości ograniczenia twardego. Ograniczenia twarde można również zmieniać, lecz procesy zwykłych użytkowników mogą je jedynie zmniejszać. Ograniczenia zasobów są dziedziczone przez podprocesy.

Priorytety procesów

Procesy uniksowe mają priorytety określające kolejność ich wykonywania w procesorze w przypadku, gdy liczba procesorów komputera jest mniejsza niż liczba gotowych do wykonywania procesów. Jądro realizuje algorytm kolejkowania procesów do wykonania, zwany schedulerem, który opiera się na priorytetach procesów, i jednocześnie może je zmieniać. Ogólnie, procesy interakcyjne zyskują wyższe priorytety, ponieważ mają związek z pracą człowieka, który nie chce czekać na reakcję komputera, ale jednocześnie pracuje z przerwami, które zwykle pozwalają na wykonywanie innych procesów.

Wszelkie modyfikacje priorytetów procesów i ingerencje w pracę schedulera są zaawansowanymi operacjami, dla zwykłych użytkowników trudnymi i niebezpiecznymi. Jednak istnieje prosty mechanizm pozwalający użytkownikowi wspomagać scheduler w wyznaczaniu priorytetów procesów. Tym mechanizmem są liczby **nice** dodawane do priorytetów. Użytkownik może uruchamiać procesy z zadaną liczbą nice (służy do tego polecenie `nice`), lub zmieniać liczbę nice wykonujących się procesów (polecenie `renice`). Co prawda zwykli użytkownicy mogą tylko zwiększać domyślną wartość nice co obniża priorytet procesu, ale można — uruchamiając w ten sposób swoje procesy „obliczeniowe" — efektywnie jakby zwiększyć priorytet procesów „interakcyjnych".

Sesja i terminal

Sesję stanowi zbiór grup procesów o wspólnym numerze sesji. Zwykle potoki poleceń uruchamiane w interpreterze poleceń stanowią oddzielne grupy procesów, lecz wszystkie należą do jednej sesji (interpretera poleceń).

Sesja może posiadać tzw. terminal sterujący. Wtedy w sesji istnieje jedna grupa procesów pierwszoplanowych (*foreground*) i dowolna liczba grup procesów drugoplanowych, czyli pracujących w tle (*background*). Procesy z grupy pierwszoplanowej otrzymują sygnały i dane z terminala.

Pojęcie terminala pochodzi od ekranowych terminali alfanumerycznych służących użytkownikom do łączenia się z komputerem. Przy połączeniach przez sieć, lub z systemu okienkowego Unix stosuje pojęcie *pseudoterminala* stanowiącego urządzenie komunikacyjne składające się z dwóch części: części użytkownika do której połączenie sieciowe wysyła dane użytkownika, i części programowej, którą widzi program na zdalnym komputerze, np. interpreter poleceń.

Mogą istnieć procesy nie posiadające terminala sterującego. Często przydatne jest by procesy pracujące ciągle w tle nie miały terminala sterującego. Takie procesy nazywa się **demonami** (ang. *daemon*).

Sygnały

- Sygnały są mechanizmem asynchronicznego powiadamiania procesów o wydarzeniach.
- Domyślną reakcją procesu na otrzymanie większości sygnałów jest natychmiastowe zakończenie pracy, czyli śmierć procesu. Oryginalnie sygnały miały służyć do sygnalizowania błędów i sytuacji nie dających się skorygować. Niektóre sygnały powodują tuż przed uśmierceniem procesu wykonanie zrzutu jego obrazu pamięci do pliku dyskowego o nazwie `core`.
- W trakcie wieloletniego rozwoju systemów uniksowych mechanizm sygnałów wykorzystywano do wielu innych celów, i wiele sygnałów nie jest już w ogóle związanych z błędami. Zatem niektóre sygnały mają inne reakcje domyślne, na przykład są domyślnie ignorowane.
- Proces może zadeklarować własną procedurę obsługi lub ignorowanie sygnału (z wyjątkiem sygnałów SIGKILL i SIGSTOP, których nie można przechwycić ani ignorować).

nr	nazwa	domyślnie	zdarzenie
1	SIGHUP	śmierć	rozłączenie terminala sterującego
2	SIGINT	śmierć	przerwanie z klawiatury (zwykle: Ctrl-C)
3	SIGQUIT	zrzut	przerwanie z klawiatury (zwykle: Ctrl-\)
4	SIGILL	zrzut	nielegalna instrukcja
5	SIGTRAP	zrzut	zatrzymanie w punkcie kontrolnym (breakpoint)
6	SIGABRT	zrzut	sygnał generowany przez funkcję abort
8	SIGFPE	zrzut	nadmiar zmiennoprzecinkowy
9	SIGKILL	śmierć	bezwarunkowe uśmiercenie procesu
10	SIGBUS	zrzut	błąd dostępu do pamięci
11	SIGSEGV	zrzut	niepoprawne odwołanie do pamięci
12	SIGSYS	zrzut	błąd wywołania funkcji systemowej
13	SIGPIPE	śmierć	błąd potoku: zapis do potoku bez odbiorcy
14	SIGALRM	śmierć	sygnał budzika (timera)
15	SIGTERM	śmierć	zakończenie procesu
16	SIGUSR1	śmierć	sygnał użytkownika
17	SIGUSR2	śmierć	sygnał użytkownika
18	SIGCHLD	ignorowany	zmiana stanu podprocesu (zatrzymany lub zakończony)
19	SIGPWR	ignorowany	przerwane zasilanie lub restart
20	SIGWINCH	ignorowany	zmiana rozmiaru okna
21	SIGURG	ignorowany	priorytetowe zdarzenie na gniazdku
22	SIGPOLL	śmierć	zdarzenie dotyczące deskryptora pliku
23	SIGSTOP	zatrzymanie	zatrzymanie procesu
24	SIGTSTP	zatrzymanie	zatrzymanie procesu przy dostępie do terminala
25	SIGCONT	ignorowany	kontynuacja procesu
26	SIGTTIN	zatrzymanie	zatrzymanie na próbie odczytu z terminala
27	SIGTTOU	zatrzymanie	zatrzymanie na próbie zapisu na terminalu
30	SIGXCPU	zrzut	przekroczenie limitu CPU
31	SIGXFSZ	zrzut	przekroczenie limitu rozmiaru pliku

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, SIGWINCH)

Sygnały mogą być wysyłane do konkretnego pojedynczego procesu, do wszystkich procesów z danej grupy procesów, albo wszystkich procesów danego użytkownika. W przypadku procesu z wieloma wątkami sygnał jest doręczany do jednego z wątków procesu.

Obsługa sygnałów

Proces może zadeklarować jedną z następujących możliwości reakcji na sygnał:

- ignorowanie
- wstrzymanie możliwości doręczenia mu sygnału na jakiś czas, po którym odbierze on wysłane w międzyczasie sygnały
- obsługa sygnału przez wyznaczoną funkcję w programie, tzw. handler
- przywrócenie domyślnej reakcji na dany sygnał

W przypadku skryptów zestaw możliwych reakcji na sygnał jest bardziej ograniczony (np. Bourne shell: polecenie trap).

Typowa procedura obsługi sygnału przez funkcję handlera może wykonać jakieś niezbędne czynności (np. skasować pliki robocze), ale ostatecznie ma do wyboru:

- zakończyć proces
- wznowić proces od miejsca przerwania, jednak niektórych funkcji systemowych nie można wznowić dokładnie od miejsca przerwania
- wznowić proces od miejsca przerwania z przekazaniem informacji przez zmienną globalną
- wznowić proces od określonego punktu

Monitorowanie procesów — program ps

Podstawowym narzędziem sprawdzania parametrów wykonujących się procesów w systemach uniksowych jest `ps`. Sprawne posługiwanie się nim wymaga pewnej nauki, ponieważ domyślne wywołanie, bez żadnych argumentów, powoduje wyświetlenie minimalnego zestawu informacji o grupie procesów uruchomionych na tym samym terminalu co `ps` (praktycznie są to zwykle: bieżący interpreter poleceń, i jego uruchomione podprocesy). Jednak `ps` pozwala precyzyjnie wybrać zarówno zestaw procesów jak i parametrów do wyświetlenia o nich.

Nabycie wprawy w użyciu `ps` jest utrudnione przez dwa niekompatybilne wcielenia tego programu: z Uniksa rodziny Systemu V i Uniksa rodziny BSD. Wersja GNU programu `ps` stara się dostosować do jednego lub drugiego stylu wywołania, odróżniając je po argumentach opcjonalnych (w wersji Systemu V zadawanych z minusem, a w wersji BSD bez minusa).

Godne uwagi proste i łatwe wywołania programu `ps`:

```
ps -ef     # szerszy zestaw informacji o wszystkich procesach
ps -el     # jeszcze szerszy zestaw informacji o wszystkich procesach
ps -fu $LOGNAME   # informacje o procesach danego uzytkownika
ps -ft pts/4      # informacje o procesach na podanym terminalu
```


Monitorowanie procesów — program top

Innym programem przydatnym do śledzenia wykonujących się procesów jest `top`

- wyświetla wybrany zestaw parametrów procesów
- wyświetla na jednym ekranie (nie przewija), okresowo go odświeżając
- z tego powodu zestaw wyświetlanych procesów jest ograniczony, i są to procesy o największym bieżacym wykorzystaniu jakichś zasobów, domyślnie CPU, posortowane od największych zasobożerców (stąd nazwa programu)
- oprócz procesów, `top` wyświetla pewne globalne statystyki systemu, takie jak liczba aktywnych procesów, wykorzystanie procesora, pamięci, itp.

`Top` jest wygodnym i często używanym programem, ale ma pewne ograniczenia. Najważniejszym jest, że nie jest programem systemowym, czyli nie znajduje się w podstawowej instalacji systemu Unix. Trzeba go sobie ściągnąć i zainstalować.

Drugim problemem jest, że `top` ma zasadniczo różne wcielenia na różnych wersjach Uniksa. Wersja na każdym systemie jest jakby trochę innym programem, o innym zachowaniu i innych możliwościach.

Wersja GNU programu `top` jest bardzo rozbudowana, ale ze względu na powyższe ograniczenia, warto uczyć się i nabierać wprawy w jej używaniu tylko jeśli jesteśmy ograniczeni do pracy na platformach Linux.

Monitorowanie pracy systemu — inne programy

Program `vmstat` zasadniczo raportuje statystyki robocze systemu pamięci wirtualnej, ale oprócz tego wyświetla pewne globalne parametry robocze systemu. Posiada tryb powtarzalny, wyświetlający jeden wiersz parametrów systemowych co określony przedział czasu.

Program `iostat` zasadniczo raportuje statystyki systemu wejścia/wyjścia urządzeń systemowych (głównie dysków, ale nie tylko) w stylu podobnym do `vmstat`, tzn. jeden wiersz parametrów, z tybem powtarzalnym.

Powyższe programy mają różne własności na różnych systemach, ale są programami systemowymi, czyli typowo znajdą się one w domyślnej instalacji. Dlatego warto pamiętać o nich, gdy mamy do czynienia z jakimś nieznanym systemem, bez dostępu administracyjnego, i chcemy uzyskać jakieś informacje o pracy systemu.

Systemy uniksowe typowo mają jeszcze więcej programów do wyświetlania statystyk roboczych systemu, niektóre dośc ciekawe, ale ich zestaw jest inny na każdym systemie. Typowo mają one nazwy kończące się na `*stat` i mieszczą się w katalogu `/usr/bin` lub `/usr/sbin`

Monitorowanie pracy systemu — system accounting

Ciekawym podsystemem pozwalającym obserwować zachowanie się całego systemu i różnych liczników statystycznych jądra jest tzw. *system accounting*. Jest realizowany za pomocą programu (`sadc`), który odczytując parametry z jądra systemu okresowo oblicza pewne statystyki chwilowe i zapisuje je na plikach w katalogu `/var/adm/sa` lub `/var/log/sysstat`. Z plików tych można następnie odczytywać długookresowe zachowanie się systemu, kreślić wykresy, itp.

`sadc`	*data collection agent* zczytuje liczniki z jądra i dopisuje do plików SA
`sar`	program użytkownika - odczytuje, interpretuje i formatuje wpisy z plików SA
`sa1`	odpalany z `cron`'a skrypt okresowo uruchamiający `sadc`
`sa2`	również odpalany z `cron`'a skrypt uruchamiający `sar` w celu obliczenia podsumowań dziennych

Pliki SA są zapisywane dziennie i typowo kasowane automatycznie po 5 dniach.

Pakiet *system accounting* jest elementem systemu Unix, ale istnieje wersja na Linuksa, dostępna często jako pakiet `sysstat`.

Process accounting

Systemy uniksowe posiadają mechanizm rejestrowania wszystkich uruchamianych procesów. Mechanizm ten, zwany *process accounting, pacct,* jest realizowany przez jądro systemu, które zapisuje na pliku `/var/adm/pacct` dla każdego procesu jego podstawowe dane: nazwę programu, użytkownika, czas startu i zakończenia, i średnie wartości zużycia zasobów (procesora i pamięci).

Włączony *process accounting* powoduje pewne obciążenie systemu, jeśli uruchamia on dużo procesów. Jednak mechanizm jest często przydatny. Wbrew swemu pierwotnemu przeznaczeniu (finansowe rozliczanie zużycia zasobów systemu), rejestrowanie wszystkich procesów przydaje się w przypadku analizowania jakichś incydentów, np. śledzenie nienormalnego zachowania programu, analizowanie włamań do systemu, itp. Jednak analizowanie zdarzeń przeszłych jest możliwe tylko wtedy gdy `pacct` było wcześniej włączone.

W obciążonych systemach pliki `pacct` przyrastają szybko. Są one okresowo odcinane i zapamiętywane ze zmienioną nazwą, a system zaczyna zapisywanie nowego pliku.

Do odczytywania treści tych plików służą programy: `acctcom`, `lastcomm`, itp. Istnieją również programy obliczające różne zestawienia i statystyki.

Process accounting na Linuksie

`/var/account/pacct`
`/var/account/pacct`
`/var/log/pacct`
	typowe lokalizacje plików `pacct`

`accton`	włączenie *process accounting* w jądrze Linuksa; należy również utworzyć pusty plik `pacct` (`pacct` pozostaje włączone po restarcie systemu)
`sa`	wyświetla różne podsumowania wykorzystania zasobów przez procesy zarejestrowane w ostatnim pliku `pacct` (wcześniejsze procesy są zarejestrowane we wcześniejszych plikach)
`lastcomm`	wyświetla zapis wszystkich procesów zarejestrowanych w ostatnim pliku `pacct`

Debugowanie pojedynczych procesów

Istnieją narzędzia do śledzenia wykonujących się procesów, nawet jeśli wykonują one program binarny. W systemie Solaris takim narzędziem jest `truss` a w systemach Linux i innych istnieje szereg narzędzi, jednym z których jest `strace`.

Programy te działają dość podobnie, i pozwalają uruchomić dowolny program, lub przyłączyć się do uruchomionego już procesu, i pokazywać w sposób symboliczny wywołania funkcji systemowych przez proces (w tym takie funkcje jak `read`/`write`/`open`), ich argumenty, wartości, itp. Pokazują również otrzymane sygnały.

System plików `/proc`

Dość wcześnie w trakcie rozwoju systemów uniksowych powstała koncepcja, by obrazy pamięci procesów były dostępne w postaci plików w katalogu `/proc`. Początkowo były to binarne pliki odpowiadające poszczególnym procesom, z czasem jednak `/proc` rozrósł się do złożonej struktury plików i katalogów.

`/proc` nie jest prawdziwym katalogiem plików dyskowych. Jego zawartość jest wirtualna i zmienia się w trakcie pracy systemu, a treść plików jest generowana lub interpretowana przez system dla każdej operacji. **Główne katalogi w tym systemie mają jako nazwy numery pid aktywnych procesów**.
W każdym katalogu procesu znajduje się zestaw plików i podkatalogów umożliwiających podgląd i modyfikowanie wybranych parametrów procesu. Prawa dostępu każdego katalogu są ustawione dla właściciela danego procesu.

Wiele programów systemowych wykorzystuje ten interface do uzyskiwania informacji o procesach. **Jednak nie ma jednoznacznie przyjętego standardu dotyczącego formatu i zawartości plików systemu /proc**. Każda wersja systemu Unix ma swoje konwencje i swój interfejs do tego systemu. Zatem nie jest obecnie możliwe pisanie w jakimkolwiek stopniu przenośnych programów ani skryptów wykorzystujących ten system.

`/proc` w systemie Solaris

W systemie Solaris katalog `/proc` przewidziany jest jako interfejs dla „silnych" narzędzi do analizowania stanu systemu i procesów. Pliki w katalogu `/proc` **odwzorowują struktury jądra** i dlatego **są plikami binarnymi**.

Podstawowe narzędzia systemu Solaris zapewniające dostęp do systemu `/proc`:

pflags	wyświetla zestaw informacji o procesie
pcred	wyświetla lub ustawia uprawnienia (identyfikację) procesu
pldd	wyświetla listę bibliotek dynamicznych zlinkowanych dla procesu
psig	wyświetla dyspozycje obsługi sygnałów i handlerów procesu
pstack	wyświetla stos wszystkich LWP (wątków) procesu
pfiles	wyświetla informacje o otwartych plikach procesu
pwdx	wyświetla katalog bieżący procesu
pstop	zatrzymuje proces
prun	ustawia proces w stan wykonywany
pwait	czeka na zakończenie procesu
ptime	uruchamia proces ze zliczaniem zużycia czasu procesora (jak polecenie time, ale bez wliczania czasu potomków)

Z `/proc` korzystają również inne narzędzia systemu Solaris, np. `dtrace`.

/proc w systemie Linux

Linux ma inną koncepcję wykorzystania mechanizmu /proc niż Solarisie. Są one raczej interfejsem użytkownika, dostępnym do manipulacji podstawowymi narzędziami systemu, niż programowym API, jak w Solarisie. Stąd pliki /proc są z reguły **plikami tekstowymi** i nie odwzorowują dokładnie struktur jądra.

Natomiast struktura podkatalogów /proc jest w Linuksie bardziej rozbudowana, i ich wykorzystanie jest posunięte dalej niż w systemach uniksowych. W szczególności, w katalogu /proc istnieją nie tylko podkatalogi poszczególnych procesów, ale również zestaw podkatalogów systemowych, pozwalających na wykonywanie niektórych operacji administracji systemem.

Trzeba zauważyć, że podobnie jak Linux jest żywym i rozwijającym się systemem, tak jego katalog /proc jest dynamicznie rozwijany i uzupełniany. Zarówno katalogi procesów, jak i katalogi systemowe, zmieniają swój skład i format plików. Dlatego, pomimo iż dostarcza on przydatnych informacji i mechanizmów, **samodzielne pisanie narzędzi je wykorzystujących jest ryzykowne**. W przyszłej wersji systemu dany plik może mieć inny format lub treść, lub może nie istnieć

/proc w systemie Linux — katalogi procesów

Podstawowe pliki w katalogu każdego procesu /proc/nnnn:

cmdline — pełny wiersz polecenia, argumenty oddzielone znakiem NUL (0)*
cwd — link do katalogu bieżącego procesu*
environ — środowisko procesu, zmienne i wartości, oddzielone zn. NUL (0)*
exe — link do programu wykonującego się procesu*
fd/ — podkatalog zawierający linki do otwartych plików procesu*
maps — lista obszarów pamięci mapowanych do plików*
mem — obraz pamięci procesu [w]
root — link do katalogu głównego procesu (ustawianego przez chroot)*
smaps — statystyki pamięci wirtualnej mapowanych obszarów pamięci (maps)
stat — zestaw informacji o procesie z jądra systemu*
statm — informacje o stanie stron pamięci procesu
status — wyciąg informacji ze stat i statm z etykietkami*
task/ — zawiera podkatalogi wątków procesu, z tymi samymi informacjami dot.wątków (jednym z nich jest katalog identyczny z katalogiem procesu)*

/proc w systemie Linux — katalogi systemowe

acpi/ — system zarządzania konfiguracją i zasilaniem (brak dokumentacji)*
bus/ — zawiera podkatalogi odpowiadające zainstalowanym magistralom
cmdline — wektor argumentów uruchomionego jądra*
cpuinfo — informacje i parametry zainstalowanego procesora/procesorów*
diskstats — statystyki operacji I/O zainstalowanych dysków
kcore — reprezentuje obraz pamięci fizycznej systemu
loadavg — statystyki obciążenia systemu i informacja o aktywnych procesach*
locks — lista blokad założonych na otwartych plikach*
meminfo — informacja o wykorzystaniu pamięci systemu*
net/ — informacje i statystyki o zainstalowanych podsystemach sieciowych*
self — to jest alias katalogu o numerycznej nazwie bieżącego procesu
stat — statystyki jądra i systemu*
sys/ — szereg zmiennych systemu, w tym parametry sieciowe sys/net/*
sysvipc/ — urządzenia komunikacji międzyprocesowej System V IPC*
uptime — czas działania systemu i całkowity czas bezczynności*
version — informacja o wersji systemu i jądra*
vmstat — statystyki podsystemu pamięci wirtualnej*

Automatyczne uruchamianie procesów: crontab

- Demon zegarowy cron uruchamia procesy w określonych dniach i godzinach według specyfikacji określonej dla każdego użytkownika. Czas uruchomienia określany jest w charakerystycznym pięciopolowym formacie:
  `minuta godzina dzien-miesiaca miesiac dzien-tygodnia`

- Program crontab umożliwia tworzenie i edycję specyfikacji zadań, tzw. **crontab**-ów. UWAGA: crontab normalnie kasuje bieżący plik specyfikacji zastępując go nowym ze stdin, czyli często pustym. Jest to częsty przypadek popsucia sobie ustawień crontaba przez początkujących administratorów.

- Bardzo ograniczone i specyficzne jest środowisko w jakim uruchamiane są procesy przez cron: interpreter /bin/sh i tylko kilka zmiennych środowiskowych, w tym bardzo ascetyczne ustawienie zmiennej PATH.

- Selektywna autoryzacja użytkowników do używania crontab (oraz at): /usr/lib/cron/{cron,at}.{allow,deny}

- Przykłady crontabów do administracji systemem:

```
/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

- Polecenie at pozwala zaprogramować uruchomienie procesu w terminie późniejszym z bardzo elastycznym językiem określeniem czasu.

- Zadanie jest uruchamiane bez terminala sterującego, z kartoteką bieżącą, środowiskiem, i ograniczeniami zasobów z momentu uruchamiania at.

- Dane wyświetlane na stdout i stderr są po zakończeniu procesu wysyłane do użytkownika pocztą elektroniczną.

- Przykład:

```
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
```


Automatyczne uruchamianie procesów: batch

- batch można uważać za skrót natychmiastowego uruchomiania procesu przez at.

- Koncepcja polecenia batch zakłada istnienie wielu kolejek zadań wsadowych wykonujących się w tle, z łatwą kontrolą ich wzajemnych priorytetów.

- Niestety, nie ma standardów, a na prawie żadnym systemie uniksowym nie ma również dokumentacji, jak te kolejki są realizowane i jak można nimi administrować. Zamiast tego wiele systemów wprowadziło własne, bardziej rozbudowane systemy kolejkowania zadań wsadowych.

- W praktyce batch przydaje się do zwykłego uruchamiania procesów w środowisku bardzo dobrze izolowanym od bieżącego interpretera poleceń, bez dostępu do terminala sterującego, współdzielenia plików stdin/stdout/stderr, itp.

Startowanie systemu

1. hardware
2. bootstrap z dysku (lub sieci)
3. jądro Uniksa 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 Uniksach 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

Włączanie użytkownika do systemu

1. `init` uruchamia proces `getty`, który inicjuje port i oczekuje na połączenie użytkownika.
2. Użytkownik dokonuje fizycznego połączenia z interfacem 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).