



Wprowadzenie do systemu UNIX

Cz. 3: Procesy

Krzysztof Boryczko

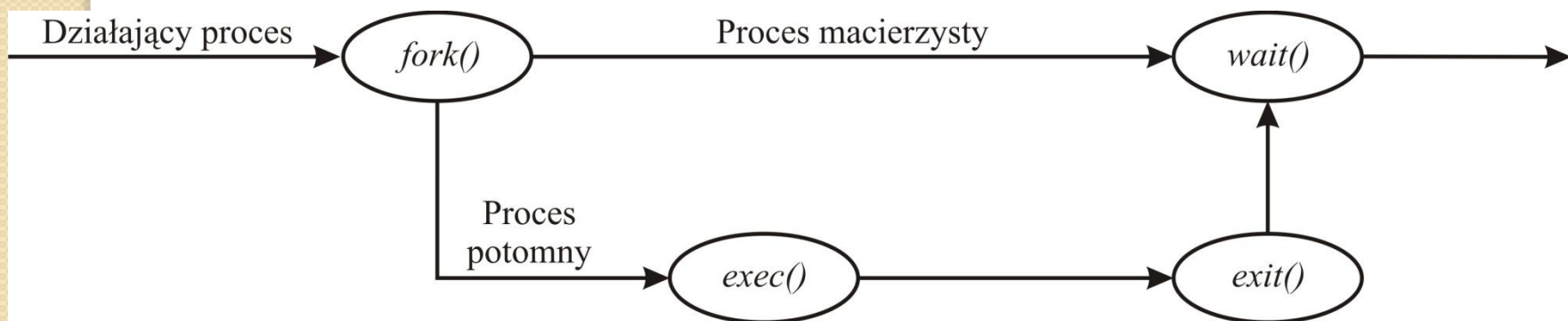
Definicja procesu

- Proces jest wykonaniem programu i składa się ze zbiorowości bajtów. System operacyjny dostrzega w nim segmenty: segment kodu, segment danych, segment BSS oraz segment(y) stos(ów).
- Wykonaniem procesów steruje jądro stwarzając wrażenie, że wiele procesów wykonuje się jednocześnie.
- Kilka procesów może być wcieleniem jednego programu.
- Proces wykonuje się przechodząc przez ściśle ustalony ciąg instrukcji stanowiący całość i nigdy nie wykonuje skoków do instrukcji innego procesu.

Opis procesu

- W kategoriach praktycznych proces w systemie UNIX jest jednostką utworzoną za pomocą funkcji systemowej *fork*, za wyjątkiem procesu o numerze 0.
- Proces wywołujący funkcję *fork* nazywa się macierzystym, a nowo utworzony potomnym.
- Każdy proces ma jeden proces macierzysty, ale może mieć wiele procesów potomnych.
- Jądro identyfikuje proces za pomocą numeru zwanego identyfikatorem procesu (PID).
- Proces o numerze 0 jest tworzony podczas inicjalizacji systemu. Tworzy on proces o numerze 1, znany jako *init*, który jest przodkiem kolejnych procesów.

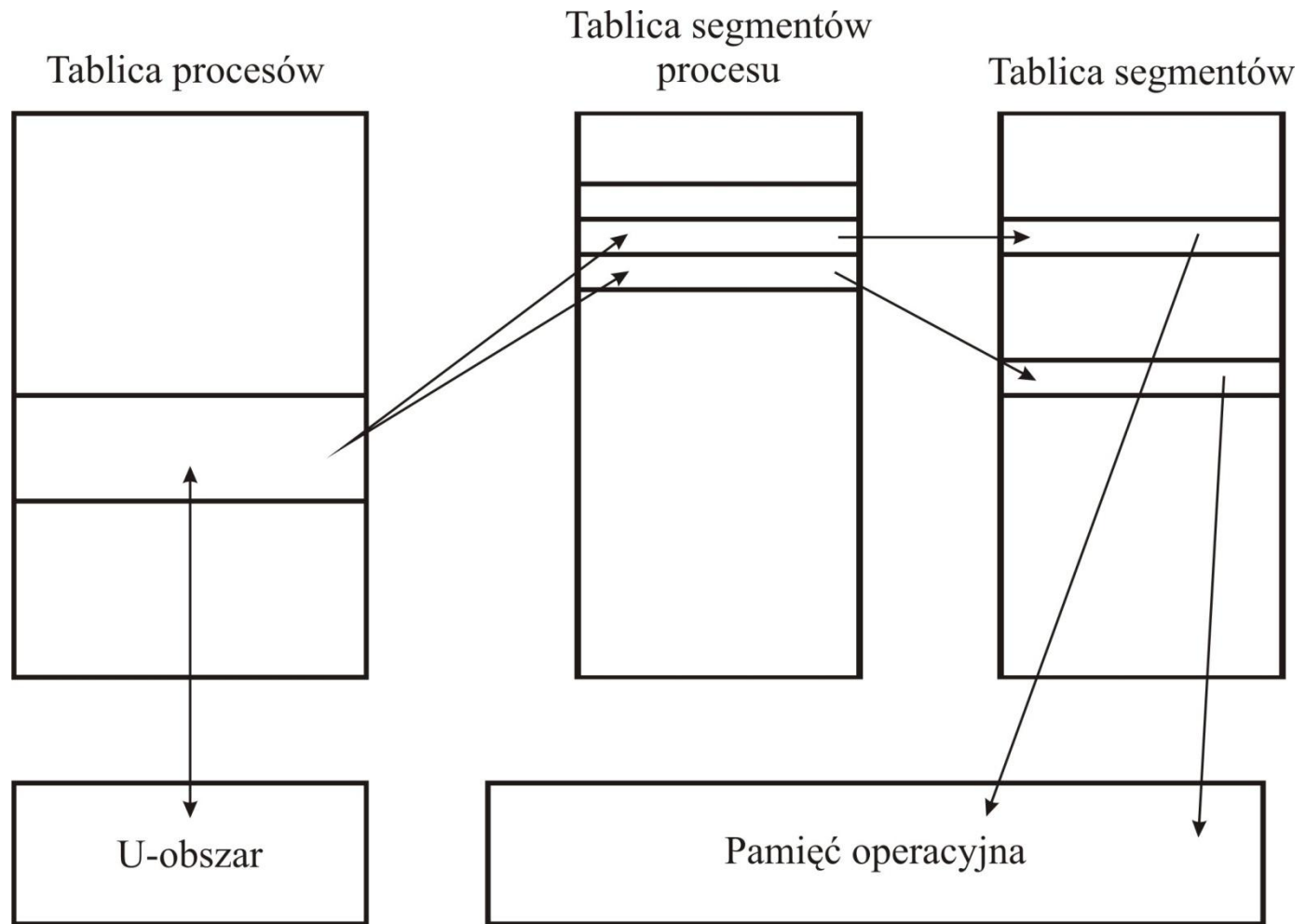
Tworzenie nowego procesu



Proces a wątek

- Wątek jest wykonaniem programu w obrębie procesu.
- Posiada licznik rozkazów, ale korzysta z segmentu kodu procesu.
- Posiada segment danych, zawierający jedynie jego dane prywatne.
- Pozostałe dane współdzieli z procesem.

Struktury opisujące proces w jądrze systemu operacyjnego



Opis struktur danych

- Jądro systemu operacyjnego przechowuje informacje o uruchomionych procesach w statycznej tablicy procesów.
- Każdy proces ma przydzielony dynamicznie tzw. *u-obszar*.
- Rekord opisujący proces w tablicy procesów zawiera wskaźnik na tablicę segmentów procesu, a te wskazania do tablicy segmentów.
- Pozycje w tablicy segmentów procesu zawierają informacje o tym co zawiera dany segment, czy jest prywatny czy współdzielony.
- W tablicy segmentów przechowywane są wskaźniki do segmentów w pamięci.

Rekord w tablicy procesów

- Pole stanu procesu.
- Pola pozwalające jądro na lokalizację segmentów procesu.
- Identyfikatory właścicieli procesu.
- Numer identyfikacyjny procesu i procesu macierzystego.
- Deskryptor zdarzenia, jeśli proces jest uśpiony.
- Parametry dla systemu szeregowania zadań do procesora.
- Pola sygnałów zawierające informacje o sygnałach wysłanych do procesu, ale jeszcze nie obsłużonych.
- Liczniki czasów wykonania procesu i wykorzystania zasobów systemu.

U-obszar

- Wskaźnik do odpowiedniego elementu tablicy procesów opisującego ten u-obszar.
- Rzeczywisty (real) i obowiązujący (effective) identyfikator użytkownika będącego właścicielem.
- Pola liczników rejestrujące czas spędzony przez dany proces i jego procesy potomne w trybie użytkownika i trybie jądra.
- Tablica obsługi sygnałów zawierająca adresy funkcji ich obsługi.
- Pole identyfikujące terminal.
- Ścieżkę dostępu do bieżącego katalogu.
- Tablicę deskryptorów plików otwartych przez proces.
- Pola opisujące graniczne wartości niektórych atrybutów procesu.

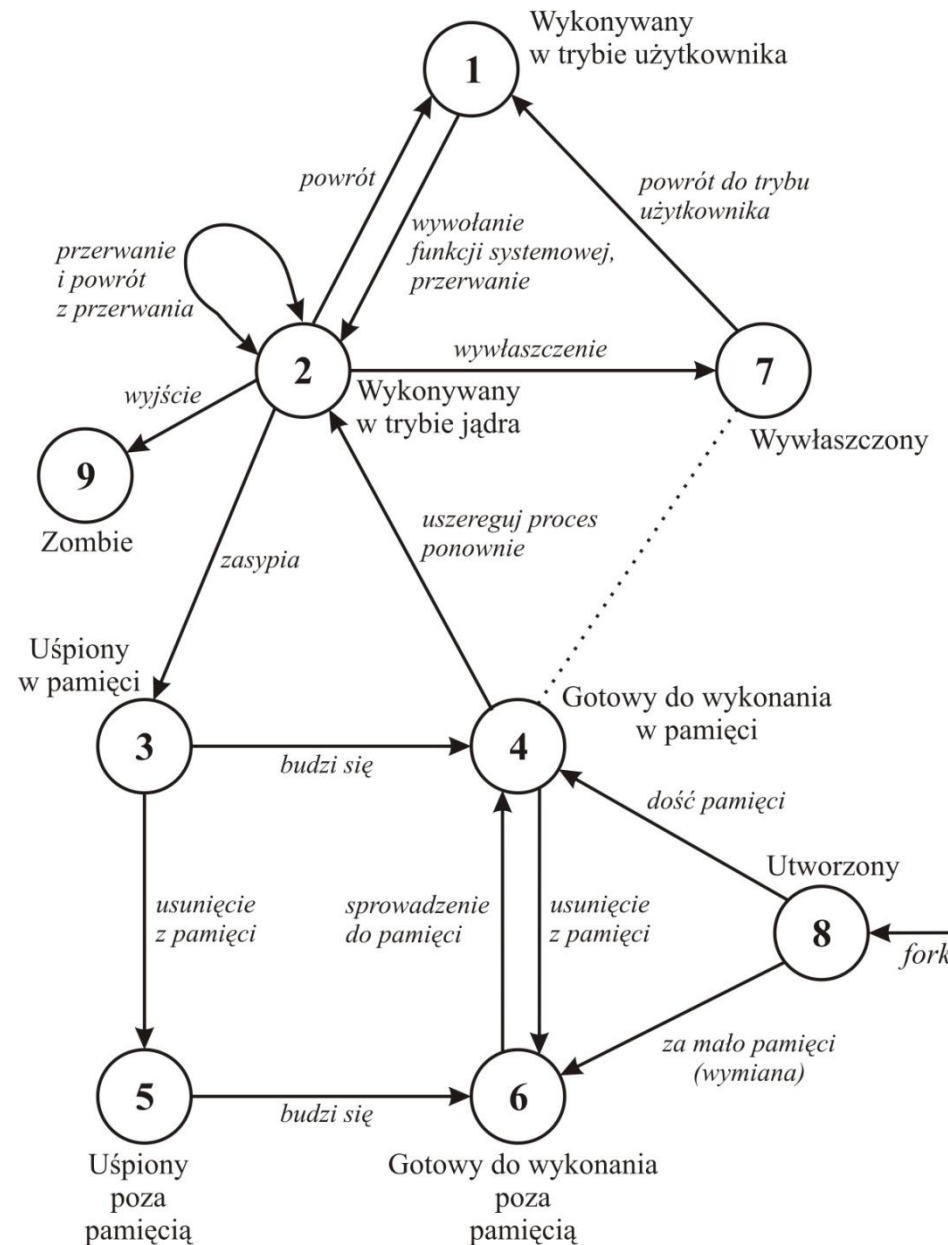
Kontekst procesu

- Kontekst to stan opisany wartościami zmiennych globalnych i struktur danych użytkownika, wartości zapisane przez niego w rejestrach procesora, wartości w polach rekordu tablicy procesów i u-obszarze oraz wartości odłożone na stosy.
- Podczas wykonywania danego procesu jądro systemu wykonuje się w kontekście tego procesu.
- Zmiana kontekstu ma miejsce wówczas, gdy zmianie ulegnie aktualnie wykonywany proces.

Diagram przejść międzystanowych

- To graf skierowany, w którym wierzchołki opisują stany zaś krawędzie zdarzenia powodujące przejście między stanami.
- Nie wszystkie stany mogą zostać praktycznie zaobserwowane ze względu na ich krótki czas trwania.

Stany procesów



Opis stanów

1. Proces wykonuje się w trybie użytkownika.
2. Proces wykonuje się w trybie jądra.
3. Proces śpi i znajduje się w pamięci operacyjnej.
4. Proces nie wykonuje się, ale znajduje się w kolejce zadań gotowych do wykonania, czekając aż jądro przydzieli mu procesor.
5. Śpi, a proces obsługi pamięci wirtualnej przesłał go do pliku wymiany (braki pamięci operacyjnej).

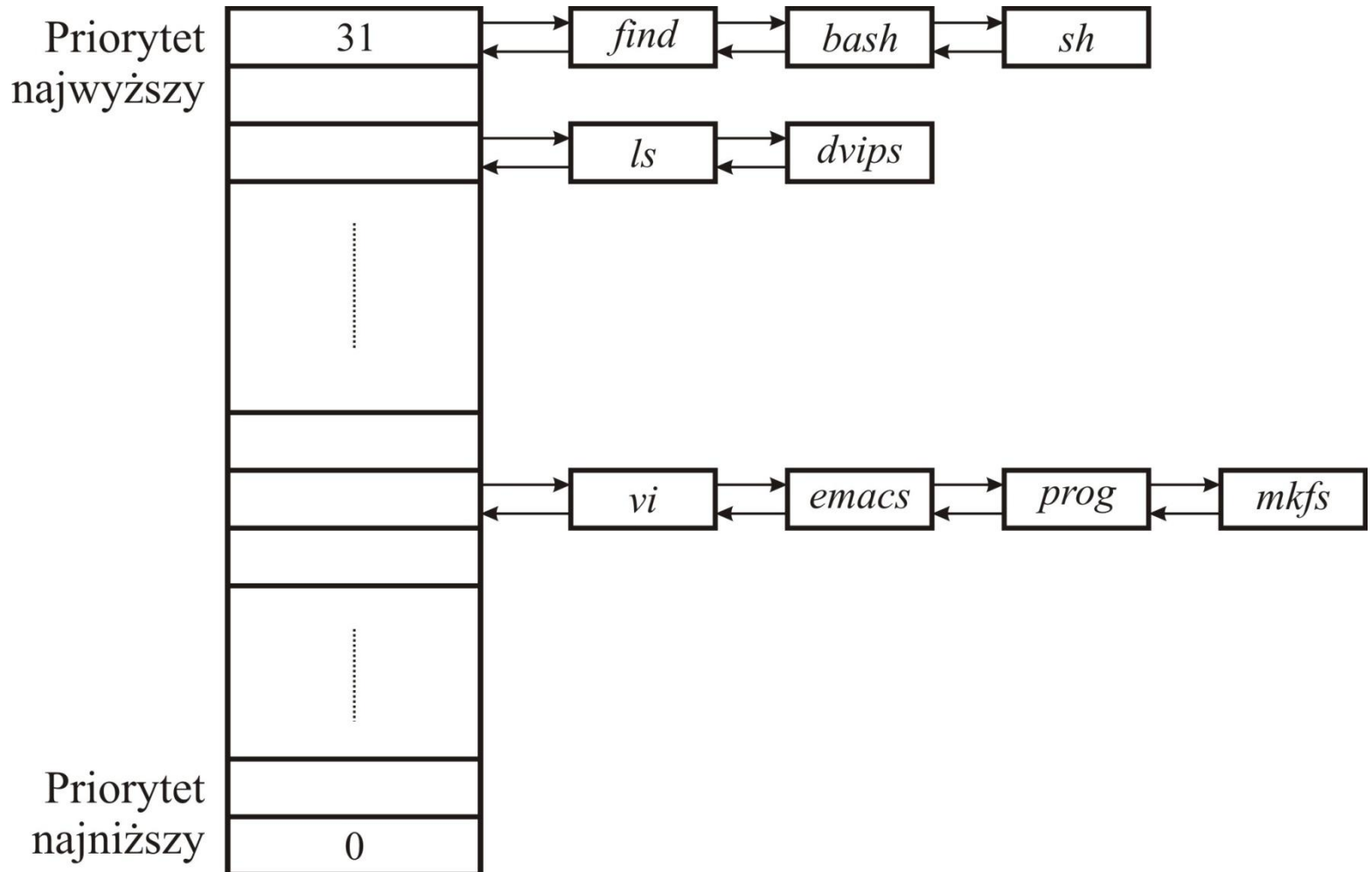
Opis stanów - cd

6. Jest gotów do wykonania lecz znajduje się w pliku wymiany (konieczność przesłania do pamięci zanim zostanie mu przydzielony procesor).
7. Przechodzi z trybu jądra do trybu użytkownika lecz jądro wywłaszcza go i przełącza kontekst aby wybrać do wykonania inny proces.
8. Jest procesem nowoutworzonym i znajduje się w stanie przejściowym.
9. Proces wykonał funkcję systemową *exit* i jest w stanie *zombie*. Segmenty i u-obszar zostały zwolnione, a pozostał rekord zawierający kod wyjścia dla procesu macierzystego.

Szeregowanie zadań w systemach BSD

- Kolejka jest zaimplementowana w postaci 32-elementowej tablicy, której każdy element będący rekordem, zawiera m.in. Wskazanie do dwukierunkowej listy procesów posiadających wartość liczbowa priorytetu z tego zakresu.
- Wykorzystywane są więc 32 kolejki przy 256 wartościach priorytetu.
- Jądro poszukuje procesu do wykonania przeglądając kolejki od góry do dołu.
- W klasycznej implementacji kwant czasu jest stały i wynosi 0.1s.
- Wyznaczanie wartości priorytetu i „ustawianie” kolejek co 1s.

Organizacja kolejki zadań w systemach BSD



Szeregowanie zadań w SVR4

- Modułarny system szeregowania zadań.
- Klasyczna implementacja udostępnia trzy procedury kolejkowania zadań: *Time Sharing* (TS), *Real Time* (RT) oraz *System* (SYS).
- Wartości atrybutów procedur kolejkowania TS oraz RT można zmieniać (*dispadmin*).
- Istnieje możliwość usunięcia istniejących i dodania własnych procedur kolejkowania.

Przykładowe procedury w SVR4

```
bash-2.05$ priocntl -l  
CONFIGURED CLASSES  
=====
```

SYS (System Class)

TS (Time Sharing)

Configured TS User Priority Range: -60 through 60

FX (Fixed priority)

Configured FX User Priority Range: 0 through 60

RT (Real Time)

Maximum Configured RT Priority: 59

IA (Interactive)

Configured IA User Priority Range: -60 through 60

Wartości priorytetów klas szeregowania w SVR4

Najwyższy priorytet globalny	159	Klasa RT
	100	
	99	Klasa SYS
Najniższy priorytet globalny	60	
	59	Klasa TS
	0	

Przeliczanie priorytetów globalnych s SVR4

- Klasa SYS obejmuje największy zakres globalnych priorytetów, zaś klasy RT oraz TS korzystają z dwóch większych segmentów. Stąd procesy czasu rzeczywistego mają priorytety wyższe niż dowolny proces klasy TS nawet wykonywany w trybie jądra).
- Istnieje jeden globalny, obejmujący wszystkie klasy zakres priorytetów.
- Klasa szeregowania jest dziedziczona od procesu macierzystego.
- Wszystkie procesy w systemie używają klasy TS, gdyż proces *init* używa tej klasy.

Zmienny kwant czasu

```
bash-2.05$ /usr/sbin/dispatchadmin -c TS -g  
# Time Sharing Dispatcher Configuration  
RES=1000
```

#	ts_quantum	ts_tqexp	ts_slpret	ts_maxwait	ts_lwait	PRIORITY	LEVEL
	200	0	50	0	50	#	0
	200	0	50	0	50	#	1
	200	0	50	0	50	#	2
	200	0	50	0	50	#	3
	200	0	50	0	50	#	4
	200	0	50	0	50	#	5
	200	0	50	0	50	#	6
	200	0	50	0	50	#	7
	200	0	50	0	50	#	8
	200	0	50	0	50	#	9
	160	0	51	0	51	#	10
	160	1	51	0	51	#	11
	160	2	51	0	51	#	12
	160	3	51	0	51	#	13

Zmienny kwant czasu

160	4	51	0	51	#	14
160	5	51	0	51	#	15
160	6	51	0	51	#	16
160	7	51	0	51	#	17
160	8	51	0	51	#	18
160	9	51	0	51	#	19
120	10	52	0	52	#	20
120	11	52	0	52	#	21
120	12	52	0	52	#	22
120	13	52	0	52	#	23
120	14	52	0	52	#	24
120	15	52	0	52	#	25
120	16	52	0	52	#	26
120	17	52	0	52	#	27
120	18	52	0	52	#	28
120	19	52	0	52	#	29
80	20	53	0	53	#	30
80	21	53	0	53	#	31
80	22	53	0	53	#	32

Zmienny kwant czasu

80	23	53	0	53	#	33
80	24	53	0	53	#	34
80	25	54	0	54	#	35
80	26	54	0	54	#	36
80	27	54	0	54	#	37
80	28	54	0	54	#	38
80	29	54	0	54	#	39
40	30	55	0	55	#	40
40	31	55	0	55	#	41
40	32	55	0	55	#	42
40	33	55	0	55	#	43
40	34	55	0	55	#	44
40	35	56	0	56	#	45
40	36	57	0	57	#	46
40	37	58	0	58	#	47
40	38	58	0	58	#	48

.....

40	48	58	0	59	#	58
20	49	59	32000	59	#	59

Zarządzanie procesami – bieżący terminal (1)

- Uruchomienie procesu: `./prog`
- Zastopowanie procesu: `^z`
- Lista procesów będących pod kontrolą bieżącego procesu logującego: *jobs*

```
boryczko@student:~/testy$ ./prog
^Z
[1]+  Stopped                  ./prog
boryczko@student:~/testy$ ./prog
^Z
[2]+  Stopped                  ./prog
boryczko@student:~/testy$ ./prog
^Z
[3]+  Stopped                  ./prog
boryczko@student:~/testy$ jobs
[1]  Stopped                  ./prog
[2]- Stopped                  ./prog
[3]+ Stopped                  ./prog
```


Zarządzanie procesami – bieżący terminal (2)

- Kontynuowanie wykonywania
zastopowanego procesu w tle – *bg %*

```
boryczko@student:~/testy$ bg %2
[2]- ./prog &
boryczko@student:~/testy$ jobs
[1]- Stopped          ./prog
[2]  Running          ./prog &
[3]+ Stopped          ./prog
boryczko@student:~/testy$
```

Zarządzanie procesami – bieżący terminal (3)

- Kontynuowanie wykonywania zastopowanego procesu na „planie pierwszym” – *fg %*
- Wiaże się z utratą wiersza polecenia. Odzyskanie:
 - Przerwanie wykonywania - *^c*
 - Można zastopować *^z* i przenieść do tła.

```
boryczko@student:~/testy$ fg %1  
./prog  
^C  
boryczko@student:~/testy$
```

Polecenie *ps*

- Te same opcje (oznaczenie literowe) w SV oraz BSD dają różny wynik działania.
- W większości systemów w poleceniu *ps* zaimplementowano działanie opcji z obu systemów. Jeśli konieczne jest działania opcji z SV to poprzedzamy ją „-”. Jeśli opcja ma działać zgodnie z BSD nie poprzedzamy jej „-”.

```
boryczko@student:~/testy$ ps -a | head -4
```

PID	TTY	TIME	CMD
3941	pts/48	00:00:00	screen
4064	pts/28	00:00:00	fishsh
4480	pts/28	00:00:01	vim

```
boryczko@student:~/testy$ ps a | head -4
```

PID	TTY	STAT	TIME	COMMAND
687	pts/32	Ss+	6:15	irssi
969	pts/14	Ss+	1:06	irssi
1218	pts/4	Ss	0:00	/bin/bash

Praktyczne zestawy opcji z SV

- -l – procesy z bieżącego terminala.
Listing w postaci długiej.

```
boryczko@student:~/testy$ ps -l
```

F S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
0 R	26029	376	25621	88	80	0	-	942	-	pts/55	00:00:03	prog
0 R	26029	388	25621	0	80	0	-	1661	-	pts/55	00:00:00	ps
0 S	26029	25621	25620	0	80	0	-	5128	-	pts/55	00:00:00	bash
0 T	26029	27309	25621	0	80	0	-	942	-	pts/55	00:00:02	prog

- -le – procesy ze wszystkich terminali.
Listing w postaci długiej.

```
boryczko@student:~/testy$ ps -le | head -5
```

F S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
4 S	0	1	0	0	80	0	-	982	?	?	00:00:41	init
1 S	0	55	1	0	80	0	-	1508	?	?	00:01:17	syslogd
5 S	0	200	1	0	80	0	-	1505	?	?	00:00:00	inetd
5 S	0	203	1	0	80	0	-	6769	?	?	00:00:48	sshd

Opis kolumn - SV

- F – flaga w jądrze systemu, wartość zależna od dystrybucji.
- S – stan procesu.
- UID – numer identyfikacyjny użytkownika będącego właścicielem procesu.
- PID – numer identyfikacyjny procesu.
- PPID – numer identyfikacyjny procesu rodzica.
- C – wykorzystanie procesora: (czas w którym procesor wykonywał instrukcje procesu)/czas wykonania) * 100.
- PRI – wartość priorytetu, dynamiczna, niemożliwa do ustalenia bezpośrednio. Im wartość liczbową niższa, tym proces jest bardziej istotny w systemie.
- NI – wartość parametru NICE, decydująca o ważności procesu w systemie. Służy do wyznaczania wartości priorytetu: $PRI = 60 + NICE + \text{kara za użycie procesora w poprzednim kolejkowaniu}$.

Opis kolumn - SV

- ADDR – numer segmentu w stosie procesu. Jeśli proces jest pod kontrolą jądra jest to adres segmentu danych procesu.
- SZ – przybliżony rozmiar w przestrzeni wymiany, konieczny do zapisania całego procesu.
- WCHAN – adres funkcji jądra podczas wykonywania której proces został uśpiony. Procesy działające oznaczane są „-”.
- TTY – nazwa terminala, z którym związany jest dany proces.
- TIME – czas procesora poświęcony na wykonywanie instrukcji danego procesu.
- CMD – krótka nazwa programu.

Ważniejsze stany

- W SV stan procesu opisany jest zawsze jedną literą.

Stan	Opis
R	Proces aktualnie wykonywany lub czeka w kolejce zadań gotowych do wykonania.
S	Proces jest uśpiony.
I	Trwa tworzenie procesu (stan pośredni).
Z	Zombie.
T	Proces jest zastopowany (^z lub działa pod kontrolą programu uruchomieniowego).
X	Rośnie i czeka na zwolnienie dodatkowej pamięci.
B	Proces czeka na dodatkową pamięć (SCO).

ps w SV - podsumowanie

- Użytkownik będący właścicielem procesu jest identyfikowany po numerze identyfikacyjnym.
- Umożliwia znajdowanie procesów macierzystych.
- Dobra informacja o wartości parametru NICE, priorytecie i wykorzystaniu procesora.
- Bardzo słaba informacja o wykorzystaniu pamięci. W szczególności nie można stwierdzić, czy dany proces znajduje się w przestrzeni wymiany, czy w pamięci operacyjnej.

Praktyczne zestawy opcji z BSD

- **au** - Listing w postaci długiej procesów skojarzonych z terminalami.

```
boryczko@student:~/testy$ ps au | head -4
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
boryczko	376	99.9	0.0	3768	348	pts/55	R	19:19	15:55	./prog
wozniczk	429	0.0	0.0	20580	2440	pts/71	Ss+	19:20	0:00	-bash
pjwal	687	0.0	0.0	42760	5584	pts/32	Ss+	Sep07	6:15	irssi

- **aux** – wszystkie procesy. Listing w postaci długiej.

```
boryczko@student:~/testy$ ps aux | head -5
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	3928	644	?	Ss	Aug26	0:41	init [3]
root	55	0.0	0.0	6032	704	?	Ss	Aug26	1:17	/usr/sbin/syslogd
root	200	0.0	0.0	6020	596	?	Ss	Aug26	0:00	/usr/sbin/inetd
root	203	0.0	0.0	27076	1140	?	Ss	Aug26	0:48	/usr/sbin/sshd

Opis kolumn - BSD

- USER – login użytkownika będącego właścicielem procesu.
- PID – numer identyfikacyjny procesu z jądra systemu operacyjnego.
- %CPU – procentowe wykorzystania czasu procesora: $(\text{czas w którym procesor wykonywał instrukcje procesu} / \text{czas wykonania}) * 100\%$
- %MEM – procentowe wykorzystanie pamięci operacyjnej systemu.
- VSZ – rozmiar wirtualny procesu (maksymalny możliwy).
- RSS – Resident Set Size – rozmiar pamięci operacyjnej zajmowanej przez proces.

Opis kolumn - BSD

- TTY – nazwa terminala, z którym związany jest proces (? – proces bez terminala).
- STAT – stan procesu.
- START – czas uruchomienia procesu.
- TIME – czas, w ciągu którego procesor wykonywał instrukcje danego procesu.
- COMMAND – polecenie, które uruchomiło proces (pełne wraz z argumentami).

Ważniejsze stany BSD (1)

- W BSD stan procesu opisany jest kilkoma znakami.

Stan	Opis
R	Proces aktualnie wykonywany lub w kolejce zadań gotowych do wykonania.
1	W systemach wieloprocessorowych – numer procesora, na którym proces się wykonuje (AIX).
S	Proces śpi od niedawna (najwyżej od 20s).
I	Proces śpi ponad 20 s.
W	Proces został przeniesiony do przestrzeni wymiany.
D	Proces uśpiony z ujemnym priorytetem (czeka na zakończenie operacji wejścia-wyjścia).
T	Proces zastopowany przez użytkownika.

Ważniejsze stany BSD (2)

Stan	Opis
N	Proces ma dodatnią wartość parametru NICE.
<	Proces ma ujemną wartość parametru NICE.
>	Proces przekroczył miękkie ograniczenie na zajętość pamięci.
Z	Zombie.
P	Proces uśpiony w oczekiwaniu na sprowadzenie strony z pamięci.
D	Proces uśpiony bez możliwości przerwania (pod kontrolą jądra).
L	Proces oczekuje na dostęp do strony w pamięci.
s	Lider sesji.
l	Proces wielowątkowy.
+	W grupie procesów wykonywane jako pierwszoplanowe.

ps w BSD - podsumowanie

- Użytkownik będący właścicielem procesu jest identyfikowany po nazwie.
- Brak informacji o PID procesu macierzystego.
- Brak precyzyjnej informacji o wartości parametru NICE. Jedynie 0/+/-.
- Precyzyjna informacja o wykorzystaniu zasobów systemowych – procesora oraz pamięci.
- Możliwość stwierdzenia, czy proces znajduje się w pamięci operacyjnej, czy w przestrzeni wymiany.

Parametr NICE

- Jego wartość mówi o ważności procesu w systemie i jest uwzględniana przy wyznaczaniu priorytetu (w różnych systemach w różny sposób).
- Jest dziedziczona od procesu macierzystego.
- W SV jego wartość jest z przedziału $[0, 40]$; 0-proces najbardziej istotny, 40-proces najmniej istotny. Wartość domyślna 20.
- W BSD jego wartość jest z przedziału $[-20, 20]$; -20-proces najbardziej istotny, 20-proces najmniej istotny. Wartość domyślna 0.

Uruchamianie procesu ze zmienioną wartością parametru NICE

- Poleceniem *nice* z opcją *-n*, której wartością jest liczba, która zostanie dodana do wartości NICE odziedziczonej po procesie macierzystym.
- Zwykły użytkownik może uruchomić proces z wartością wyższą od domyślnej.
- Użytkownik *root* może używać dowolnych wartości parametru NICE z określonego przedziału.

```
boryczko@student:~/testy$ ps -l
```

F S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
0 S	26029	26742	26741	0	80	0	- 5099	-	pts/55	00:00:00		bash
0 R	26029	27215	26742	0	80	0	- 1661	-	pts/55	00:00:00		ps

```
boryczko@student:~/testy$ nice -n+7 ./prog &  
[1] 27224
```

```
boryczko@student:~/testy$ ps -l
```

F S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
0 S	26029	26742	26741	0	80	0	- 5099	-	pts/55	00:00:00		bash
0 R	26029	27224	26742	98	87	7	- 942	-	pts/55	00:00:06		prog
0 R	26029	27279	26742	0	80	0	- 1661	-	pts/55	00:00:00		ps

```
boryczko@student:~/testy$ nice -n-7 ./prog &  
[2] 27834
```

```
boryczko@student:~/testy$ nice: cannot set niceness: Permission denied
```

```
[2]+ Exit 1          nice -n-7 ./prog
```

```
boryczko@student:~/testy$
```


Zmiana wartości parametru NICE działającego procesu

- Programem *renice*, podając docelową wartość parametru jako argument.
- Zwykły użytkownik może:
 - W systemach rygorystycznych jedynie zwiększać wartość parametru.
 - W systemach liberalnych zwiększać i zmniejszać ale do wartości domyślnej.
- Polecenie posiada dwie użyteczne opcje:
 1. -p – umożliwia podanie PID procesów, których wartość parametru NICE ma zostać zmieniona. Jest to opcja domyślna.
 2. -u – umożliwia podanie nazwy lub UID użytkownika, wszystkim procesom którego zostanie zmieniona wartość parametru NICE.

```
boryczko@student:~/testy$ renice 12 -p 27224
```

```
27224: old priority 7, new priority 12
```

```
boryczko@student:~/testy$ ps -l
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
0	S	26029	26742	26741	0	80	0	-	5099	-	pts/55	00:00:00	bash
0	R	26029	27224	26742	99	92	12	-	942	-	pts/55	00:03:26	prog
0	R	26029	28813	26742	0	80	0	-	1661	-	pts/55	00:00:00	ps

```
boryczko@student:~/testy$ renice 5 -p 27224
```

```
renice: 27224: setpriority: Permission denied
```

```
boryczko@student:~/testy$
```

Interpretacja praktyczna wartości parametru NICE

```
boryczko@student:~/testy$ nice -n+5 ./prog &
```

```
[1] 29677
```

```
boryczko@student:~/testy$ ./prog &
```

```
[2] 29684
```

```
boryczko@student:~/testy$ ps -l
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
0	S	26029	26742	26741	0	80	0	- 5099	-	pts/55	00:00:00		bash
0	R	26029	29677	26742	99	85	5	- 942	-	pts/55	00:00:10		prog
0	R	26029	29684	26742	99	80	0	- 942	-	pts/55	00:00:22		prog
0	R	26029	30127	26742	0	80	0	- 1661	-	pts/55	00:00:00		ps

```
boryczko@student:~/testy$ ps -l
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
0	S	26029	26742	26741	0	80	0	- 5099	-	pts/55	00:00:00		bash
0	R	26029	29677	26742	99	85	5	- 942	-	pts/55	00:01:10		prog
0	R	26029	29684	26742	99	80	0	- 942	-	pts/55	00:02:31		prog
0	R	26029	30127	26742	0	80	0	- 1661	-	pts/55	00:00:00		ps

Sterowanie wykonaniem procesów

- Najprościej poprzez wysyłanie sygnałów.
- Sygnał jest informacją wysyłaną asynchronicznie; nadawca przechodzi do wykonywania dalszych instrukcji nie czekając na „efekt” działania sygnału.
- Do wysyłania sygnałów do procesów służy polecenie *kill*.

Polecenie *kill* (1)

- Lista dostępnych sygnałów: *kill -l*
- Ważniejsze sygnały:
 - 15 TERM – polecenie zakończenia wykonywania procesu.
 - 9 KILL – natychmiastowe zakończenie wykonywania procesu.
 - 3 QUIT – ^c.
 - 20 TSTP - ^z.
 - 19 STOP – zastopowanie procesu.
 - 18 CONT – kontynuowanie wykonywania zastopowanego procesu.

```
boryczko@student:~/testy$ 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
```

Polecenie kill (2)

- Użycie wymaga podania nazwy lub numeru sygnału oraz numer(ów) procesów, do których ma on zostać wysłany.
- Domyślnym sygnałem jest sygnał 15 TERM.
- Przykłady:
 - `kill -s SIGSTOP 12456 %1`
 - `kill -s STOP 12456 %1`
 - `kill -STOP 12456 %1`
 - `kill -19 12456 %1`

Demon atd

- Daje możliwość jednokrotnego uruchamiania zadań o określonym czasie z dokładnością do jednej minuty.
- Uruchamiany z plików: */etc/rc.d/init.d* lub */etc/init.d*
- Sprawdzenie działania: *service atd status*
- Uruchomienie: *service atd start*
- Zapewnienie uruchomienia podczas startu systemu: *chkconfig -a atd*
- Działanie: o pełnej minucie sprawdza zawartość katalogu */var/spool/at* i jeśli znajdzie zadanie do uruchomienia w danym momencie czasu, to podejmuje próbę uruchomienia.

Jednokrotne uruchamianie procesów

- Oprócz działania `atd` wymaga konfiguracji w plikach `/etc/at.allow` oraz `/etc/at.deny`
- Plik `/etc/at.allow` jest plikiem tekstowym, zawierającym w jednej linii nazwę użytkownika (user name), któremu wolno korzystać z polecenia `at`.
- Plik `/etc/at.deny` jest plikiem tekstowym, zawierającym w jednej linii nazwę użytkownika (user name), któremu NIE wolno korzystać z polecenia `at`.
- Najpierw sprawdzany jest plik `at.allow` i jeśli istnieje, to wszystkim użytkownikom w nim zdefiniowanym wolno korzystać z polecenia `at`.
- Jeśli `at.allow` nie istnieje, to sprawdzany jest plik `at.deny` i wszystkim użytkownikom, którzy nie są w nim zdefiniowani wolno korzystać z polecenia `at`.
- Jeśli nie istnieją `at.allow` oraz `at.deny` to tylko użytkownik `root` może korzystać z polecenia `at`.
- Istnienie pustego pliku `at.allow` powoduje, że żaden użytkownik oprócz użytkownika `root` nie może korzystać z polecenia `at` (niezależnie od istnienia pliku `at.deny`).
- Istnienie pustego pliku `at.deny` i brak pliku `at.allow` powoduje, że każdy użytkownik może korzystać z polecenia `at`.

Polecenie *at*

- Służy do umieszczania zadań do wykonania o określonym czasie w kolejce zadań.
- Czas specyfikujemy najczęściej jako przedział od znacznika czasu, np. now, noon, ... , lub podając dokładną datę i godzinę:
 - at now + 2 min
 - at -t 201701061342

```
boryczko@student:~/testy$ at now + 2 min
warning: commands will be executed using /bin/sh
at> ./prog
at> <EOT>
job 128 at Sun Nov  4 19:44:00 2012
boryczko@student:~/testy$ at now + 10 min
warning: commands will be executed using /bin/sh
at> ./prog
at> <EOT>
job 129 at Sun Nov  4 19:53:00 2012
```


Zarządzanie kolejką zadań

- Polecenie *atq* wypisuje zadania w kolejce bieżącego użytkownika, zarówno wykonywane jak i oczekujące na uruchomienie.

```
boryczko@student:~/testy$ atq
128    Sun Nov  4 19:44:00 2012 a boryczko
129    Sun Nov  4 19:53:00 2012 a boryczko
boryczko@student:~/testy$ atq
128    Sun Nov  4 19:44:00 2012 = boryczko
129    Sun Nov  4 19:53:00 2012 a boryczko
```

- Polecenie *atrm* usuwa zadania z kolejki, zarówno wykonywane jak i oczekujące na uruchomienie. Obowiązują prawa własności.

```
boryczko@student:~/testy$ atrm 128
Warning: deleting running job
boryczko@student:~/testy$ atrm 129
```

Demon crond

- Uruchamiany z plików: */etc/rc.d/init.d* lub */etc/init.d*
- Sprawdzenie działania: `service crond status`
- Uruchomienie: `service crond start`
- Zapewnienie uruchomienia podczas startu systemu: `chkconfig --a crond`
- Działanie: o pełnej minucie sprawdza zawartość katalogów: */var/spool/cron* i jeśli znajdzie zadanie do uruchomienia w danym momencie czasu, to podejmuje próbę uruchomienia.
- Daje możliwość cyklicznego uruchamiania zadań w oparciu o zawartość pliku zwanego tablicą *crona*.

Cykliczne uruchamianie procesów

- Oprócz działania `crond` wymaga konfiguracji w plikach `/etc/cron.allow` oraz `/etc/cron.deny`
- Plik `/etc/cron.allow` jest plikiem tekstowym, zawierającym w jednej linii nazwę użytkownika (user name), któremu wolno korzystać z polecenia `crontab`.
- Plik `/etc/cron.deny` jest plikiem tekstowym, zawierającym w jednej linii nazwę użytkownika (user name), któremu NIE wolno korzystać z polecenia `crontab`.
- Najpierw sprawdzany jest plik `cron.allow` i jeśli istnieje, to wszystkim użytkownikom w nim zdefiniowanym wolno korzystać z polecenia `crontab`.
- Jeśli `cron.allow` nie istnieje, to sprawdzany jest plik `cron.deny` i wszystkim użytkownikom, którzy nie są w nim zdefiniowani wolno korzystać z polecenia `crontab`.
- Jeśli nie istnieją `cron.allow` oraz `cron.deny` to tylko użytkownik `root` może korzystać z polecenia `crontab`.
- Istnienie pustego pliku `cron.allow` powoduje, że żaden użytkownik oprócz użytkownika `root` nie może korzystać z polecenia `crontab` (niezależnie od istnienia pliku `cron.deny`).
- Istnienie pustego pliku `cron.deny` i brak pliku `cron.allow` powoduje, że każdy użytkownik może korzystać z polecenia `crontab`.

Polecenie *crontab*

- Każdy użytkownik może mieć dokładnie jedną tablicę crona. Jest to plik tekstowy: */var/spool/cron/login_name*
- Do zarządzania tablicą crona służy polecenie *crontab*. Podstawowe opcje:
 - -e – uruchamia edytor i umożliwia edycję pliku.
 - -r – usuwa tablice crona bieżącego użytkownika.
 - -l – listuje tablicę crona bieżącego użytkownika.

Budowa tablicy crona

- Plik tekstowy zawierający w jednej linii specyfikację jednego zadania do uruchomienia. Znak # to komentarz od wystąpienia do końca linii.
- Każda linia składa się z 6-ciu kolumn oddzielonych co najmniej jednym białym znakiem:
 1. minuta uruchomienia, zakres 0-59
 2. godzina uruchomienia, zakres 0-23
 3. dzień uruchomienia, zakres 1-31
 4. miesiąc uruchomienia, zakres 1-12
 5. dzień tygodnia, zakres 0-7 (kalendarz kolejarski: 0 i 7 niedziela, 1 poniedziałek, 2 wtorek, itd.)
 6. polecenie do uruchomienia (najbezpieczniej bezwzględna ścieżka dostępu)
- W kolumnach 1-5 możliwa jest:
 - specyfikacja kilku możliwości, np.: 1,7,9
 - specyfikacja zakresu możliwości, np.: 1-12,20-31
 - specyfikacja „co ile”, np.: 1-31/2 lub 1-31/4
 - specyfikacja każdy z zakresu, np.: */2

Tablica crona - przykład

przykładowa tablica crona

#	minuta	godzina	dzien	miesiac	dz. tyg.	Polecenie
0		4	*	10-12,1-7	5	/root/mkbckup
1-31/2		17	2-30/2	10,11,12	*	date >> /tmp/date
* * * *	*					mail -s „Przypomnienie” wyz@pcz.edu.pl< text

dalsze zadania do cyklicznej realizacji

Słowo o badaniu wydajności systemów uniksowych



Idea

- Polega na wyznaczaniu stanu systemu komputerowego w ściśle określonych momentach czasowych.
- Interwały pomiarów:
 - Przy badaniu wydajności procesora 2-5 sekund.
 - Przy badaniu wydajności układów wejścia/wyjścia 20-50 sekund.
- Obowiązuje zasada nieoznaczoności Heisenberga: obserwacja zjawiska zaburza jego naturalny przebieg. W tym wypadku chodzi o konieczność uruchomienia procesu monitorującego, zatem system zachowuje się inaczej.

Programy do monitorowania

- BSD

- top – ogólne dane o systemie i wykorzystaniu jego zasobów przez procesy.
- vmstat – ogólne dane statystyczne o działaniu systemu, obciążeniu procesora, pamięci i dysków.
- iostat – wejście-wyjście dysku i terminali oraz obciążenie procesora.
- ps – stan procesów.
- w – krótka charakterystyka procesów poszczególnych użytkowników.
- uptime – obciążenie procesora i ogólny stan systemu.
- pstat – rozmiary i wykorzystanie tablic systemowych.

- SystemV

- top – ogólne dane o systemie i wykorzystaniu jego zasobów przez procesy.
- sar – ogólne dane statystyczne o działaniu systemu, obciążeniu procesora, pamięci i dysków.
- ps – stan procesów.

Open Monitoring Distribution (OMD)

- Modułowy system do monitorowania zasobów systemów informatycznych.
- Bazuje na Nagiosie.
- Oprócz gotowych „zestawów” monitorujących umożliwia tworzenie własnych przez implementacje wtyczek (język przypomina C).
- Dostępny w postaci kodu źródłowego lub pakietów instalacyjnych dla większości systemów linuksowych.

OMD

CheckMK

2013.06.28

2013.06.28

mk (admin) 11:43

Tactical Overview

Hosts

Problems

Unhandled

0

Services

Problems

Unhandled

3

Quicksearch

Views

▼ Dashboards

Main Overview

Network Topology

► Hosts

► Hostgroups

► Services

► Servicegroups

▼ Business Intelligence

All Aggregations

Hostname Aggregations

Problem Aggregations

Single-Host Aggregations

Single-Host Problems

► Problems

▼ Addons

Search Graphs

▼ Other

Comments

Downtimes

Host- and Service events

Host- and Service notifications

Search Global Logfile

EDIT

Speed-O-Meter

Bookmarks

New rule Delay service notifications

Service Interface 2, Eiger

Add Bookmark

WATO - Configuration

Main Menu

Hosts & Folders

Host Tags

Global Settings

Host & Service Parameters

Host Groups

Service Groups

Users & Contacts

Roles & Permissions

Contact Groups

Time Periods

Logfile Pattern Analyzer

BI - Business Intelligence

Distributed Monitoring

Audit Logfile

Host Statistics

Up

12

Down

0

Unreachable

0

In Downtime

0

Total

12

Service Statistics

OK

338

In Downtime

1

On Down host

0

Warning

1

Unknown

0

Critical

2

Total

342

Host Problems (unhandled)

state

Host

Icons

Age

Status detail

Service Problems (unhandled)

State

Host

Service

Icons

Status detail

Age

CRIT

Eiger

Job Sicherung

CRIT -

Exit-Code: 1

CRIT

Started:

2013-07-02

20:00:01,

Real-Time: 0

sec,

User-Time: 0

sec,

System-Time:

0 sec,

Filesystem

Reads: 96,

Filesystem

Writes: 16,

Max.

Memory:

5.65MB, Avg.

Memory:

0.00B, Vol.

Context

Switches: 23,

Invol. Context

Switches: 8

2013-06-28 16:27

CRIT

Lastrechner

Emails_of_sales

CRIT - 4

emails in

inbox (4

current, 0

new)

21 min

WARN

Lastrechner

Check_MK inventory

WARNING -

2 unchecked

services

(mounts:2)

29 min

Events of recent 4 hours

Time

Host

Service

Check output

29 min

Lastrechner

Check_MK inventory

WARNING - 2 unchecked services (mounts:2)

29 min

Lastrechner

Check_MK inventory

WARNING - 2 unchecked services (mounts:2)

29 min

Lastrechner

Check_MK inventory

WARNING - 2 unchecked services (mounts:2)

29 min

Lastrechner

Check_MK inventory

WARNING - 2 unchecked services (mounts:2)

31 min

Lastrechner

OMD_Builds_rh61-64

OK - 3 Pakete:
omd-2013.07.03.dmmk-rh61-31.x86_64.rpm
omd-2013.07.03.mk-rh61-31.x86_64.rpm
omd-2013.07.03.dmmk-rh61-31.x86_64.rpm

31 min

Lastrechner

OMD_Builds_rh61-64

OK - 3 Pakete:
omd-2013.07.03.dmmk-rh61-31.x86_64.rpm
omd-2013.07.03.mk-rh61-31.x86_64.rpm
omd-2013.07.03.dmmk-rh61-31.x86_64.rpm

31 min

Lastrechner

OMD_Builds_rh61-64

OK - 3 Pakete:
omd-2013.07.03.dmmk-rh61-31.x86_64.rpm
omd-2013.07.03.mk-rh61-31.x86_64.rpm
omd-2013.07.03.dmmk-rh61-31.x86_64.rpm

31 min

Lastrechner

OMD_Builds_rh61-64

OK - 3 Pakete:
omd-2013.07.03.dmmk-rh61-31.x86_64.rpm
omd-2013.07.03.mk-rh61-31.x86_64.rpm
omd-2013.07.03.dmmk-rh61-31.x86_64.rpm

31 min

Lastrechner

OMD_Builds_rh61-32

OK - 3 Pakete:
omd-2013.07.03.dmmk-rh61-31.i386.rpm
omd-2013.07.03.mk-rh61-31.i386.rpm
omd-2013.07.03.dmmk-rh61-31.i386.rpm

31 min

Lastrechner

OMD_Builds_rh61-32

OK - 3 Pakete:
omd-2013.07.03.dmmk-rh61-31.i386.rpm
omd-2013.07.03.mk-rh61-31.i386.rpm
omd-2013.07.03.dmmk-rh61-31.i386.rpm

© Mathias Kettner

Uwagi do monitorowania

- Klasyczne programy: vmstat, iostat, sar uruchamia się z dwoma argumentami: interwał oraz liczba powtórzeń.
- Jeśli w systemie dochodzi do stronicowania lub wymiany, to obciąża ona systemu wejścia-wyjścia. Nie musi to być wynikiem działania procesu.