Akademia Techniczno-Humanistyczna w Bielsku-Białej

LABORATORIUM

Obliczeń Równoległych i Systemów Rozproszonych

Sprawozdanie nr 1

Zarządzanie procesami

GRUPA: 2B / SEMESTR: 5 / ROK: 3

Igor Gawłowicz / 59096

Krystian Niedźwiedź / 58824

Proces

to ciąg (sekwencja) logicznie uporządkowanych czynności, w wyniku których powstaje określony efekt (rezultat) działania (produkt, usługa), z którego korzysta klient (zewnętrzny lub wewnętrzny).

Każdy proces może utworzyć jedene lub więcej procesów potomnych **(child)** wobec którego staje się procesem macierzystym **(parent)**. W chwili tworzenia procesu system operacyjny alokuje, celem jego reprezentacji, strukturę danych w postaci **PCB** (Process Control Block)

Każdy system operacyjny oferuje usługi umożliwiające pobranie informacji o aktywności i stanie bieżących procesów. W systemach rodziny ***POSIX*** służą temu m.in. zestaw poleceń konsoli:

`ps [option] -o [format]`

np.

`ps group users tty 3 -o pid,cmd`

Wyświetli dla grupy *users* z terminala *3* informację o jej procesach podając *PID* oraz komendę jaka uaktywniła proces.

np. zwykłe `ps` zwróci nam:

```
$ ps
    PID TTY          TIME CMD
    575 pts/2    00:00:00 bash
   3140 pts/2    00:00:00 nano
   3714 pts/2    00:00:00 ps
```

Listing procesów

Procesy możemy wyświetlić w postaci struktury drzewa, poczynając od procesu *init* albo *pid*

`pstree [options] [pid|user]`

np zwykłe `pstree` bez żadnych argumentów zwróci nam:

```
$ pstree
pause─┬─dockerd─┬─containerd───8*[{containerd}]
      │         └─10*[{dockerd}]
      ├─logger
      ├─python─┬─editor-proxy───4*[{editor-proxy}]
      │        └─sudo───tmux-agent───4*[{tmux-agent}]
      ├─rsyslogd───3*[{rsyslogd}]
      ├─sshd─┬─sshd───sshd───bash───bash
      │      └─sshd───sshd───bash───start-shell.sh───tmux
      └─tmux───bash───pstree
```

Ponieważ informacja odnośnie identyfikacji procesów ma kluczowe znaczenie przy zarządzaniu nimi, każde API systemowe daje nam możliwość w jakiś sposób uzyskać takie informacje. np w języku c/c++

```c
#include <stdio.h>
#include <unistd.h>
int main(void)
{
    printf("Currnet ID\t%d\n", (int)getpid());
    printf("Parent ID\t%d\n", (int)getppid());

    return 0;
}
```

Taki program zwróci nam informacje o identyfikatorze obecnego procesu i identyfikatorze procesu macierzystego.

```
$ gcc pid.c -o pid
$ ./pid
Current ID      1670
Parent ID       575
```

Nieco inne możliwości śledzenia daje nam komenda `top`, która jest szczególnie ważna w przypadku konieczności monitorowania pracy komputera jako węzła cluster'a.

```
top - 10:03:14 up 18 min,  2 users,  load average: 0.34, 0.48, 0.37
Tasks:  30 total,   1 running,  29 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :  16002.5 total,  14164.4 free,    375.4 used,   1462.8 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  15335.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    330 root      20   0 1356272  42540  30556 S   0.3   0.3   0:02.26 containerd
      1 65535     20   0     972      4      0 S   0.0   0.0   0:00.00 pause
      7 root      20   0 1238212  15992   9828 S   0.0   0.1   0:01.17 gateway
     18 root      20   0    3896   3040   2748 S   0.0   0.0   0:00.04 bash
     25 root      20   0   88652  10496   6172 S   0.0   0.1   0:01.34 python
     33 root      20   0  220796   2840   1852 S   0.0   0.0   0:00.99 rsyslogd
     34 root      20   0 1678536  13596   7676 S   0.0   0.1   0:00.62 command-
recorde
```

```c
#include <stdio.h>
int main(int argc, char **argv)
{
    int i;
    for(i=0; i < argc; i++)
    {
        printf("%4d:... %s\n", i, argv[i]);
    }
    return 0
}
```

Powyższy program możemy wykonać podając wraz z poleceniem uruchamiającym argumenty, które ma podać funkcji main.

```
$ ./child pierwszy drugi trzeci
    0:... ./child
    1:... pierwszy
    2:... drugi
    3:... trzeci
```

Możemy także wykonać taki program z poziomu innego programu dla którego dany program stanie się wtedy procesem macierzystym.

```c
#include <stdio.h>
#include <unistd.h>
int main(void)
{
    char *arg1="pierwszy", char *arg2="drugi", char *arg3="trzeci";
    prinf("- wywołanie (samobójcze) potomka ----------\n");
    execl("./child", arg1, arg2, arg3, '\0');

    return 0
}
```

```
$ ./parent
- wywołanie (samobójcze) potomka ----------
    0:... pierwszy
    1:... drugi
    2:... trzeci
```

W powyższym programie egzekwujemy program `child` podając mu argumenty po czym za pomocą `'\0'` dajemy programowi znać że to jest koniec argumentów.

Za pomocą funkcji `fork()` możemy podzielić program na rodzica i potomka co wykorzstane w zły sposób może prowadzić do nieoczekiwanych rezultatów, ponieważ jeśli nie sprawimy żeby program poczekał na zakończenie procesu potomka wynik może się nam rozjechać ponieważ oba procesy będą chciały wykonywać się równocześnie, a nie zawsze każdy z nich będzie jednocześnie szybko się wykonywał.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main(void)
{
    int status;

    switch(fork())
    {
        case -1: //W razie błędu dla Parent
        printf("<parent> oj niedobrze, niedobrze\n");
        break;
        case 0: //Child
        printf("<child> pozdrowienia dla potomka\n");
        break;
        default: //Parent
        printf("<parent> ja jestem PARENT\n");
        wait(&status);
        printf("<parent> potomek skończył, zwrócił: :%d\n", status);
    }

    return 0;
}
```

Taki program zwróci nam następujący komunikat:

```
$ ./fork
<parent> ja jestem PARENT
<child> pozdrowienia dla potomka
<parent> potomek skończył, zwrócił: :0
```

Wnioski

Podczas analizy i eksploracji tematu zarządzania procesami w systemach operacyjnych, zrozumialiśmy, że procesy są fundamentalnymi jednostkami, które umożliwiają współbieżność i wielozadaniowość w systemach komputerowych. Poznaliśmy istotne koncepcje, takie jak identyfikatory procesów, tworzenie procesów potomnych i komunikację między nimi. Warto zdobyta wiedza o zarządzaniu procesami stanowi istotną podstawę w programowaniu systemowym i tworzeniu oprogramowania na platformy Unix.