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 top watch time
```

```
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.

Listing procesów

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

```
pstree [options] [pid|user]
```

Zarządzanie procesami

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)
{
    prinf("Currnet ID\t%d\n", (int)getpid());
    prinf("Parent ID\t%d\n", (int)getppid());

    return 0
}
```

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

```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
        print("<parent> ja jestem PARENT\n");
        wait(&status);
        printf("<parent> potomek skończył, zwrócił: :%d\n", status);
    }

    return 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.