WOJSKOWA AKADEMIA TECHNICZNA

Wydział Cybernetyki

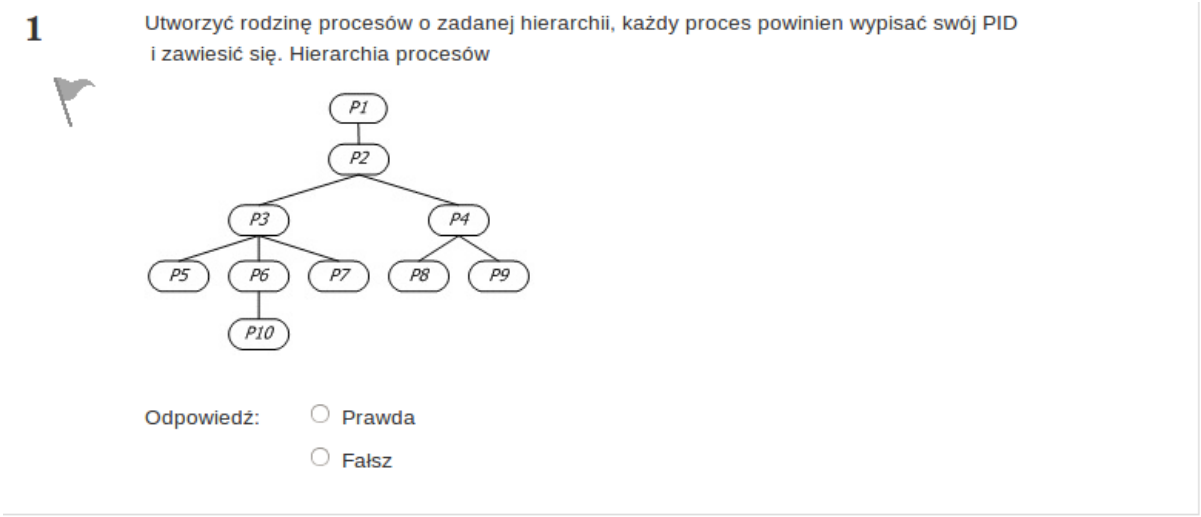
Systemy Operacyjne

„Procesy w systemach UNIX/LINUX”.

Wykonał: kpr. pchor. Damian Krata
Grupa: I4X3S1 numer na zajęciach 6
Data wykonania ćwiczenia: 10.11.2015r.
Prowadzący: mgr inż. Łukasz Laszko

Zadanie:

Utworzyć rodzinę procesów o zadanej hierarchii, każdy proces powinien wypisać swój PID i zawiesić się. Hierarchia procesów:



Rozwiązanie postawionego problemu:

Zgodnie z definicją podaną podczas wykładu, proces należy rozumieć jako obiekt systemu operacyjnego, który wykonuje program i udostępnia mu środowisko wykonania (przestrzeń adresową oraz punkt(y) sterowania). Ma on przydzielone zasoby typu pamięć, procesor itp. Części z nich jest do jego wyłącznej dyspozycji, a część jest współdzielona z innymi procesami, np. segment kodu w przypadku współbieżnego wykonywania tego samego programu w ramach kilku procesów. W zależności od aktualnie posiadanych zasobów wyróżnia się stany procesu (np. wykonywany, uśpiony, gotowy), które zmieniają się cyklicznie w związku z wykonywanym programem lub ze zdarzeniami zachodzącymi w systemie.

W zakresie obsługi procesów, system UNIX udostępnia mechanizm tworzenia nowych procesów, usuwania procesów oraz uruchamiania programów. Każdy proces, z wyjątkiem procesu systemowego o identyfikatorze 1, tworzony jest przez inny proces, który staje się jego przodkiem, zwanym też procesem rodzicielskim (rodzicem). Nowo utworzony proces nazywany jest potomkiem lub procesem potomnym. Procesy w systemie UNIX tworzą zatem drzewiastą strukturę hierarchiczną, podobnie jak katalogi.

Przejdźmy zatem do realizacji zadania. Do mojego programu dodałem trzy biblioteki. Przy czym zwrócę uwagę na dwie ostatnie. Otóż, w plikach sys/types.h(typy danych) oraz unistd.h(niezbędne funkcje dla systemów POSIX) zdefiniowane są funkcje służące do obsługi procesów opisanych w 2 i 3 części pomocy systemowej. Dalej, z wykorzystaniem funkcji typu get – , został zwrócony identyfikator danego **getpid()** procesu, w tym przypadku macierzystego. W dalszej części, dla procesów potomnych, przy użyciu funkcji , podaję również PID rodzica aktualnego procesu. **getppid()**

Potomek tworzony jest w wyniku wywołania przez przodka funkcji systemowej **fork().** Po jej wywołaniu, tworzony jest nowy proces, który współdzieli z procesem macierzystym obszar kodu, ale ma odrębny obszar danych. Funkcja ta zwraca wartość 0 dla procesu potomnego, a procesowi macierzystemu zwraca identyfikator potomka (PID). Po utworzeniu, potomek kontynuuje wykonywanie programu swojego przodka od miejsca wywołania funkcji **fork(),** by następnie zakończyć swe działanie.

Kod programu:

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main(void)

{

	printf("Macierzysty >PID: %d\n",getpid());

	if(fork()==0) //P2

	{

		printf("P2: PID:%d, PPID:%d\n", getpid(), getppid());

		if(fork()==0) //P3

		{

			printf("P3: PID:%d, PPID:%d\n", getpid(), getppid());
```

```
if(fork()==0) //P5
{
        printf("P5: PID:%d, PPID:%d\n", getpid(), getppid());
        pause();
        return 0;
}
if(fork()==0) //P6
{
        printf("P6: PID:%d, PPID:%d\n", getpid(), getppid());

        if(fork()==0) //P10
        {
                printf("P10: PID:%d, PPID:%d\n", getpid(), getppid());
                pause();
                return 0;
        }
        pause();
        return 0;
}
if(fork()==0) //P7
{
        printf("P7: PID:%d, PPID:%d\n", getpid(), getppid());
        pause();
```

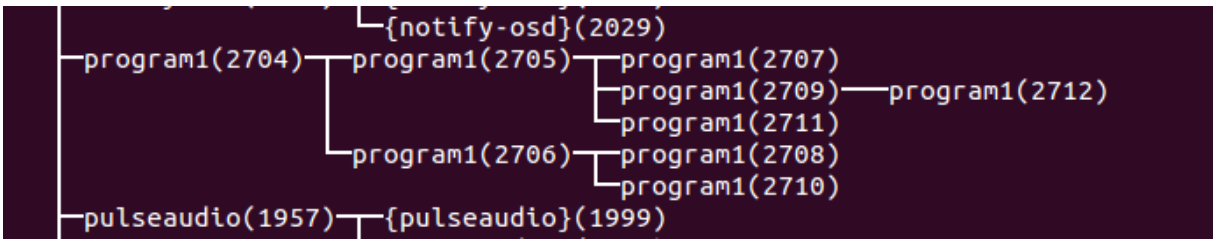
```
            return 0;
        }

        pause();
        return 0;
}



if(fork()==0) //P4
{
        printf("P4: PID:%d, PPID:%d\n", getpid(), getppid());

        if(fork()==0) //P8
        {
            printf("P8: PID:%d, PPID:%d\n", getpid(), getppid());
            pause();
            return 0;
        }
        if(fork()==0) //P9
        {
            printf("P9: PID:%d, PPID:%d\n", getpid(), getppid());
            pause();
            return 0;
```

```
                }


                pause();

                return 0;

        }


        pause();

        return 0;

    }


}
```

Wynik działania programu:

Poniżej przedstawiam zrzuty ekranów obrazujące wynik działania programu.

1. Drzewo procesów pokazane za pomocą instrukcji pstre –cpl



2. Wynik działania programu i drzewo procesów dla instrukcji
   pstre –p | grep ”\(bash\program1\)” | grep –v grep

Porównując, identyfikatory poszczególnych procesów czy relacje między nimi wyświetlone poprzez pstree, a informacje o tych identyfikatorach w ramach działania mojego programu stwierdzam, ze zadanie zostało zrealizowane poprawnie.

Wnioski:

Cel ćwiczenia laboratoryjnego został osiągnięty w stopniu bardzo dobrym. W trakcie jego realizacji miałem możliwość praktycznego wykorzystania wiedzy dotyczącej procesów – ich tworzenia, grupowania w drzewa itp. Rezultat pracy w łatwy sposób mogłem sprawdzić wywołując polecenie pstre, które jest wizualnym odpowiednikiem polecenia ps. Przedstawienie utworzonych procesów w takiej formie jest przejrzyste i pozwoliło na stuprocentowe stwierdzenie poprawności kodu programu.