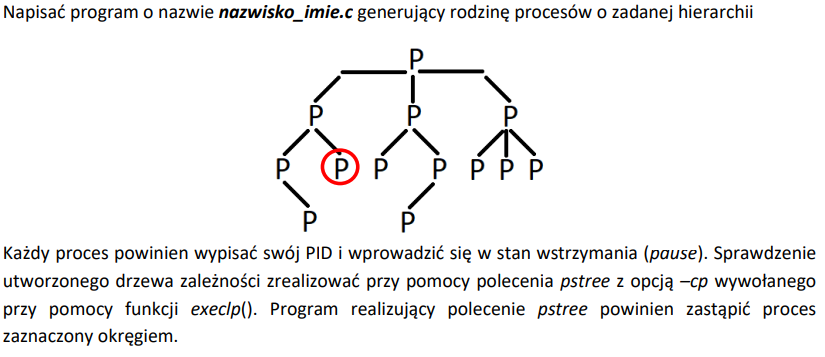
|  |  |  |
| --- | --- | --- |
| |  | | --- | | **INSTYTUT TELEINFORMATYKI I AUTOMATYKI**  **Wydział Cybernetyki WAT** | | |
| |  | | --- | | Przedmiot: SYSTEMY OPERACYJNE  SPRAWOZDANIE Z ĆWICZENIA LABORATORYJNEGO NR 9 | | |
| **Temat**: Procesy | |
| **Wykonał:**  xxxxx | **Data wykonania ćwiczenia:**  xxxxx  **Prowadzący ćwiczenie:**  xxxxx |

1. **Treść zadania laboratoryjnego**



1. **Opis rozwiązania, komentarze, wnioski**

Program:

#include<stdio.h>

#include<unistd.h>

int main(void)

{

    int x,y,z;

    char p[20];

    sprintf(p,"%d",getpid());

    printf("P0: %d PPID: %d\n", getpid(),getppid());

    if((x=fork())==0){

        printf("P1: %d PPID: %d\n", getpid(),getppid());

        if(fork()==0){

            printf("P1.1: %d PPID: %d\n", getpid(),getppid());

            if(fork()==0){

            printf("P1.1.1: %d PPID: %d\n", getpid(),getppid());

            }

        }

        else if(fork()==0){

            printf("P1.2: %d PPID: %d\n", getpid(),getppid());

            sleep(1);

            execlp("pstree","pstree","-cp",p,(char\*)NULL);

        }

    }

    else if((y=fork())==0){

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

        if(fork()==0){

            printf("P2.1: %d PPID: %d\n", getpid(),getppid());

        }

        else if(fork()==0){

            printf("P2.2: %d PPID: %d\n", getpid(),getppid());

            if(fork()==0){

            printf("P2.2.1: %d PPID: %d\n", getpid(),getppid());

            }

        }

    }

    else if((z=fork())==0){

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

        if(fork()==0){

            printf("P3.1: %d PPID: %d\n", getpid(),getppid());

        }

        else if(fork()==0){

            printf("P3.2: %d PPID: %d\n", getpid(),getppid());

        }

        else if(fork()==0){

            printf("P3.2: %d PPID: %d\n", getpid(),getppid());

        }

    }

    pause();

    return 0;

}

Konstrukcja funkcji sprintf:

sprintf(tablica, typ\_zmiennej, wartości);

Konstrukcja funkcji execlp:

execlp(ścieżka, lista\_argumentów, NULL);

Program wypisuje procesy wszystkie na raz, więc kolejność nie jest zgodna z kolejnością w programie.

W moim programie występował błąd przy wypisywaniu procesów, który rozwiązałem dając polecenie sleep(1). Spowodowało to, że funkcja execlp wykonuje się z opóźnieniem 1 sekundy, dlatego widać go na samym końcu kompilaji.

1. **Wyniki uruchomienia programu**

Obraz zawierający tekst

Opis wygenerowany automatycznie