Systemy Operacyjne – laboratorium Zachodniopomorska Szkoła Biznesu			
Grupa/Specjalność	**Imię i nazwisko**	**Data wykonania (rrrr.mm.dd)**	**Tryb studiów**
Game Design	**Sylwester Dawidowicz**	**9.05.2020**	**Z**
Nr laboratorium	**Temat**		
3	Przetwarzanie sygnałów w systemie Linux		

Na tych laboratoriach zadaniem było przygotowanie programu tworzącego procesy potomne (fork) oraz ich ubijanie (kończenie). Główne założenia i wygląd programu:

1) Definiujemy funkcję tworzącą procesy potomne:

```
int *forks(int forks_amount)
{
        int *fork_list = malloc(forks_amount * sizeof(int));

            for (int i = 0; i < forks_amount; i++)
                {
                    fork_list[i] = fork();
                    if (fork_list[i] == 0)
                        {
                                for (;;)
                                    {
                                        //infinite loop
                                    }
                        }
                }

            return fork_list;
    }
```

Procesy te będą wykonywać nieskończoną pętlę **for**.

2) Definiujemy funkcję zabijającą (kończącą) proces:

```
void printfork_list(int *fork_list, int fork_amount)
    {
        for (int i = 0; i < fork_amount; i++)
            {
                if (fork_list[i] >= 0)
                    {
                        printf("%d. Fork [ParentPID: %d  |  ChildrenPID: %d]\n", i, getppid(), fork_list[i]);
                    }
                else
                    {
                        printf("%d. Proces jest martwy\n", i);
                    }
            }

    }
```

3) Kolejna funkcja jest odpowiedzialna za wyświetlanie utworzonych forków:

```
void printfork_list(int *fork_list, int fork_amount)
    {
        for (int i = 0; i < fork_amount; i++)
            {
                    if (fork_list[i] >= 0)
                        {
                                printf("%d. Fork [ParentPID: %d  |  ChildrenPID: %d]\n", i, getppid(), fork_list[i]);
                        }
                    else
                        {
                                printf("%d. Proces jest martwy\n", i);
                        }
             }

    }
```

4) W funkcji main() realizujemy nasze funkcje poprzez wybieranie procesów do ubicia:

```
int main(int argc, char *argv[])
    {
        if (argv[1] == NULL)
            {
                    fprintf(stderr, "Podaj liczbę ");
                    exit(EXIT_FAILURE);
                }

            int amount = atoi(argv[1]);
        int *fork_list = forks(amount);
        int option;

            while (1)
                {
                    printfork_list(fork_list, amount);
                printf("Który proces potomny chcesz zabic? Podaj numer procesu: ");
                    scanf("%d", &option);
                if (option >= 0 && option < amount)
                    {
                            int result = kill(fork_list[option], SIGKILL);
                            if (result >= 0)
                                {
                                printf("Zabito proces o numerze pid: '%d', zatrzymano metoda: %d\n", fork_list[option], SIGKILL);
                                    fork_list[option] = SIGKILL * -1;
                                }
                            else
                                {
                                    perror("Nie mozna zabic tego procesu potomnego!");
                                    printf("\n");
                                }
                        }   else
                                {
                                        fprintf(stderr, "Nieprawidlowy indeks procesu!\n");
                                }

                    kill_that_proccess(fork_list, amount);
                }
    }
```

Podsumowując, laboratoria sprawiły mi średnią trudność Kończenie procesów potomnych pozwala nam w łatwy sposób zarządzań programem oraz pamięcią.