3 Procesy

3.1 Funkcja `fork`

1. Wykonaj program wywołujący następujące funkcje:

```
printf("Poczatek\n");
fork();
printf("Koniec\n");
```

2. Dodaj wywołanie funkcji bibliotecznej `sleep` za wywołaniem funkcji `fork`. Po uruchomieniu programu w tle, sprawdź komendą `ps -l` listę procesów w systemie. Zwróć uwagę na identyfikatory procesów macierzystych (kolumna PPID).

```
# ./a.out &
# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1011  2567  2566  0  75   0 -  1233 -      pts/2    00:00:00 bash
0 S  1011  2589  2567  0  75   0 -   337 schedu pts/2    00:00:00 a.out
0 S  1011  2590  2589  0  75   0 -   337 schedu pts/2    00:00:00 a.out
1 R  1011  2591  2567  0  75   0 -   905 -      pts/2    00:00:00 ps
```

3. Sprawdź wartość zwracaną przez funkcję `fork`. Wyświetl wartości PID i PPID korzystając z funkcji systemowych `getpid` i `getppid`. Przykładowa implementacja:

```
int x;
x = fork();
printf("wynik fork=%d  PID=%d  PPID=%d\n", x, getpid(), getppid());
```

4. Ile procesów powstanie po wykonaniu następującego fragmentu kodu:

```
fork();
fork();
```

   A ile dla tego fragmentu:

```
if (fork()==0) {
  fork();
}
```

3.2 Funkcja exec

1. Wykonaj program wywołujący następujące funkcje:

```
printf("Poczatek\n");
execl("/bin/ls", "ls", "-l", NULL);
printf("Koniec\n");
```

2. Wykonaj polecenie `ls -l` w procesie potomnym:

```
printf("Poczatek\n");
if (fork()==0)
{
  /* proces potomny */
  execlp("ls", "ls", "-l", NULL);
  exit(1);
}
printf("Koniec\n");
```

3. Wprowadź oczekiwanie na zakończenie procesu potomnego w procesie macierzystym:

```
if (fork()==0)
{
  /* proces potomny */
  ...
}
else
{
  /* proces macierzysty */
  wait(NULL);
  printf("Koniec\n");
}
```

4. Odczytaj status zakończenia procesu potomnego:

```
if (fork()==0)
{
  /* proces potomny */
  exit(5);
}
else
{
  /* proces macierzysty */
  int stat;
  wait(&stat);
  printf("status=%d\n", stat>>8);
}
```

5. Zaobserwuj powstawanie procesów *zombie* dodając oczekiwanie funkcją `sleep` w procesie macierzystym.
6. Zapoznaj się z innymi wersjami funkcji `exec`, np. `execvp`. Napisz program, który wykona następujące polecenie korzystając z funkcji `execvp`:

```
sort -n -k 3,3 -t: /etc/passwd
```

3.3 Przekierowania strumieni standardowych

1. Wykonaj polecenie `ps ax` zapisując jego wynik do pliku `out.txt`. Oto przykładowy fragment implementacji:

```
int fd;
close(1);
fd = open("out.txt", O_WRONLY | O_CREAT, 0644);
execlp("ps", "ps", "ax", NULL);
```

2. Ustaw deskryptory plików na właściwych pozycjach w tablicy otwartych plików korzystając z funkcji systemowej `dup` lub `dup2`:

```
int fd;
fd = open("out.txt", O_WRONLY | O_CREAT, 0644);
dup2(fd, 1);
```

3. Zrealizuj programowo następujące zlecenie:

```
# ls -l -a -i >> wynik.txt
```

4. Zrealizuj programowo następujące zlecenie:

```
# grep xy < dane.txt > wynik1.txt 2> wynik2.txt
```

5. Napisz program który wykona zlecenie powłoki takie jak w poprzednim zadaniu, ale dodatkowo wypisze na ekranie komunikat przed i po wykonaniu zlecenia.