Temat zajęć	Procesy w systemie Linux
Zakres materiału	Tworzenie i wykonywanie procesów w systemie Linux

Materiał teoretyczny

- funkcje *fork(), wait()* i funkcje z rodziny *exec..()*
- analiza argumentów linii poleceń i kodu zakończenia procesu

Treść zadania

Napisać języku C program spełniający poniższe wymagania:

- program akceptuje dokładnie jeden argument wywołania – uruchomienie go z inną liczbą argumentów albo bez argumentów ma skutkować wyprowadzeniem na *stderr* informacji o błędzie i przerwaniem pracy z kodem zakończenia równym 1;
- otrzymany argument ma być liczbą naturalną – jeśli tak nie jest, program wyprowadza na *stderr* informację o błędzie i kończy pracę z kodem zakończenia równym 2;
- liczba otrzymana w argumencie ma pochodzić z przedziału <1..13> - w przeciwnym przypadku program wyprowadza na *stderr* informację o błędzie i kończy pracę z kodem zakończenia równym 3;
- jeśli otrzymany argument jest równy 1 lub 2, program od razu kończy pracę z kodem zakończenia 1;
- w przeciwnym przypadku program uruchamia dwa procesy potomne (funkcja *fork()*);
- każde z dzieci zastępuje swój kod (wybrana funkcja z rodziny *exec..()*, np. *execv()* albo *execl()*), uruchamiając ten sam program, który jest wykonywany przez rodzica i przekazuje mu jako argument odpowiednio: do pierwszego dziecka – liczbę o jeden mniejszą od otrzymanej, do drugiego dziecka – o dwa mniejszą od otrzymanej;
- rodzic czeka na zakończenie obu procesów potomnych, a następnie wypisuje na *stdout* trzy linie tekstu:
  1. swój PID, PID pierwszego zakończonego dziecka, argument, z którym pierwsze dziecko zostało uruchomione i kod powrotu pierwszego dziecka;
  2. swój PID, PID drugiego zakończonego dziecka, argument, z którym drugie dziecko zostało uruchomione i kod powrotu drugiego dziecka;
  3. swój PID i sumę kodów powrotów obu dzieci (sumę należy wyprowadzić w tej samej kolumnie, co kody powrotu potomków);
- rodzic kończy pracę, zwracając kod powrotu równy sumie kodów powrotów obu dzieci;
- w efekcie w ostatniej linii wyjścia powinna znaleźć się wartość tego wyrazu ciągu Fibonacciego, którego numer podano jako argument uruchomienia programu.

Uwaga: poprawna sekwencja operacji to *fork-fork-wait-wait* (czyli rodzic **najpierw** uruchamia **dwa** procesy potomne, a potem czeka na zakończenie obu); konstrukcja kodu, w której operacje układają się w ciąg *fork-wait-fork-wait* (procesy potomne pracują po kolei) dyskwalifikuje rozwiązanie!

Przykładowe wyjście z programu:

```
$ ./prog5 5
 99827  99830     2   1
 99827  99831     1   1
 99827                2

 99828  99832     2   1
 99828  99833     1   1
 99828                2

 99826  99828     3   2
 99826  99829     2   1
 99826                3

 99825  99826     4   3
 99825  99827     3   2
 99825                5
```

Uwaga! Kod źródłowy programu (1 plik) po oddaniu prowadzącemu zajęcia laboratoryjne musi zostać jako **załącznik** przesłany na adres `son1@wi.zut.edu.pl`:

- plik z kodem źródłowym musi mieć nazwę: `numer_indeksu.so.lab05.c` (np. `66666.so.lab06.c`),
- plik musi zostać wysłany z poczty uczelnianej (domena `zut.edu.pl`),
- temat maila musi mieć postać: `SO IN1 99X LAB05`
  gdzie *99X* to numer grupy laboratoryjnej (np. `SO IN1 20A LAB05`),
- w pierwszych trzech liniach kodu źródłowego w komentarzach (każda linia komentowana osobno) musi znaleźć się:
  - informacja identyczna z zamieszczoną w temacie maila,
  - imię i nazwisko osoby wysyłającej maila,
  - adres e-mail, z którego wysłano wiadomość
    np.:

```
// SO IN1 20A LAB05
// Jan Nowak
// nj66666@zut.edu.pl
```

- e-mail nie może zawierać żadnej treści (tylko załącznik).

Dostarczone kody programów będą analizowane pod kątem wykrywania plagiatów. Niewysłanie wiadomości, wysłanie jej w formie niezgodnej z powyższymi wymaganiami lub wysłanie pliku, który nie będzie się kompilował i uruchamiał, będzie traktowane jako brak programu i skutkowało otrzymaniem za niego oceny niedostatecznej.