*W wyniku wykonania poniższych zadań powinien powstać **program**, którego kody źródłowe powinny zostać przesłane na adres antyplagiatu (informacja w ramce na końcu zadania).*

W rodzinie funkcji **wait** znajdziemy m.in. funkcje systemowe **wait3** i **wait4**. Działają one analogicznie do, odpowiednio, funkcji **wait** i **waitpid**. Zwracają jednak dodatkowo rozszerzoną informację o zasobach zużytych przez zakończony proces potomny. Korzystają m.in. z jednej tych funkcję napisz własną wersję polecenia **time**. W dalszej części instrukcji o naszej wersji polecenia **time** będziemy mówili **program**, natomiast o programie, którego czas wykonania będziemy badali: **program testowy**. Wymagania dotyczące programu:

- przyjmuje jako argumenty nazwę programu testowego do wykonania (wraz z listą argumentów programu testowego, jeżeli ich wymaga);
- przyjmuje opcję, której wystąpienie spowoduje, że wyniki działania programu testowego będą wyświetlane na ekran (np. **-v**); domyślnie wyniki te, zarówno kierowane na **stdout** jak i na **stderr**, mają być ukryte;
- przyjmuje opcję, która określi ile razy ma zostać wykonany program testowy (bez podania opcji domyślnie jeden raz);
- w wyniku działania programu wyświetlona zostanie informacja o czasach wykonania programu testowego:
    - rzeczywistym (pomiar ze "stoperem", przybliżony, od momentu uruchomienia programu testowego do jego zakończenia, np. z użyciem funkcja **clock_gettime**);
    - użytkownika (ile czasu proces wykonywał się w przestrzeni użytkownika);
    - systemowym (ile czasu proces wykonywał się w przestrzeni jądra);
- jeżeli program testowy będzie wywoływany więcej niż raz (podaliśmy odpowiednią opcję) to poza pojedynczymi wynikami pomiarów na końcu powinny pojawić się wartości średnie (real, user, system).

Do prezentacji przygotuj proste programy testowe, które zademonstrują działanie programu gdy mamy przewagę obliczeń w przestrzeni użytkownika albo w przestrzeni jądra.

Przykładowe wywołanie programu może wyglądać następująco (20 razy uruchamiamy program testowy **find /etc**, wyświetlane są wyniki działania programu testowego, na końcu wyświetlanych jest 20 pomiarów czasu i czasy uśrednione):

```
./myTime -v -t 20 find /etc
```

Podpowiedź: w celu przekierowania standardowego wyjścia do pliku /dev/null (z poziomu procesu) można wykorzystać poniższy fragment kodu:

```
close(1);
int h = open("/dev/null", O_WRONLY);
dup2(h, 1);
```

Przed wysłaniem pliku źródłowego na antyplagiat jego nazwę zmieniamy na: ***numer_indeksu.ps.lab04.main.c*** *(czyli np. 66666.ps.lab04.main.c).*

Kody źródłowe po oddaniu prowadzącemu zajęcia laboratoryjne muszą zostać jako załączniki przesłane na adres **pss1@zut.edu.pl** (wysyłamy jeden mail z czterema załącznikami):

- pliki z kodami źródłowymi muszą mieć nazwę zgodne ze wzorcem podanym w treści zadania,
- mail musi zostać wysłany z poczty uczelnianej (domena **zut.edu.pl**),
- temat maila musi mieć postać: **PS IS1 999X LAB04**, gdzie 999X to numer grupy laboratoryjnej (np. PS IS1 321 LAB04),
- 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.:

      ```
      // PS IS1 321 LAB04
      // Jan Nowak
      // nj66666@zut.edu.pl
      ```

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

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.