Zadanie - zestaw 3

Tworzenie procesów. Środowisko procesu, sterowanie procesami.

Zadanie 1. Drzewo procesów (15%)

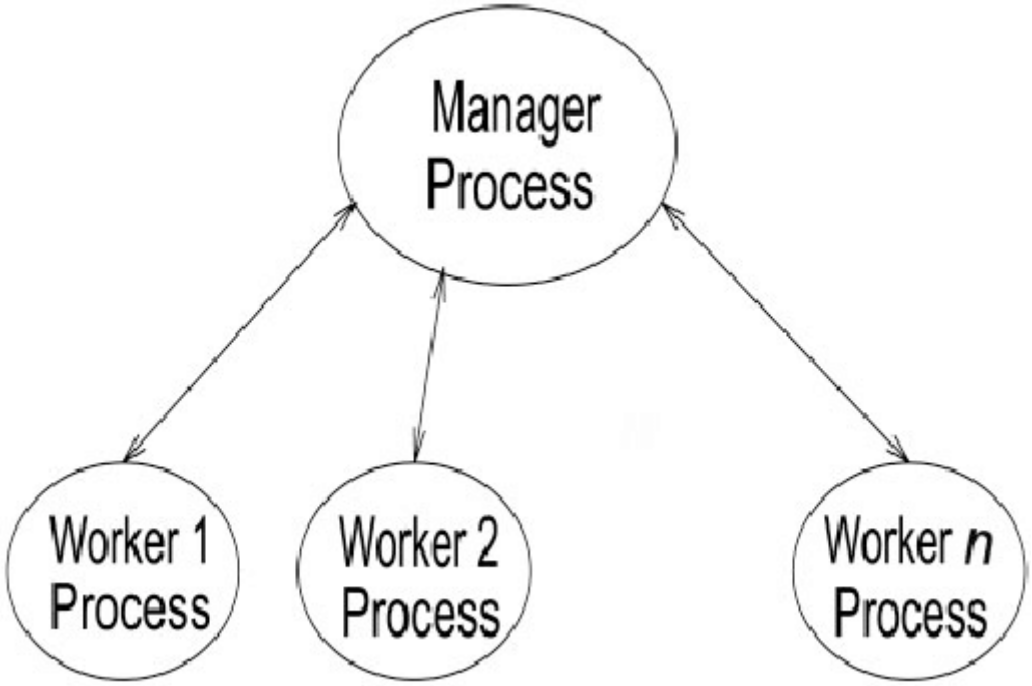
Modyfikując zadanie 2 z poprzedniego zestawu, napisz program, który dla każdego z podkatalogów utworzy proces potomny i wywoła polecenie `ls -l`. Wynik `ls` poprzedź wypisaniem ścieżki względnej od katalogu podanego jako argument oraz numeru `PID` procesu odpowiedzialnego za przeglądanie określonego poziomu.

Zadanie 2. Równoległe mnożenie macierzy (50%)

Napisz program *macierz* do równoległego mnożenia macierzy — zawartość każdej z macierzy (wejściowej lub wynikowej) znajduje się w osobnych plikach. Argumenty operacji mnożenia są treścią pliku `lista`, będącego pierwszym argumentem wywołania programu. Plik `lista` zawiera, w pojedynczej linii:

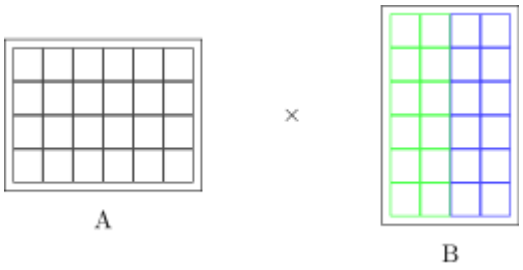
- Nazwę pliku z pierwszą macierzą wejściową.
- Nazwę pliku z drugą macierzą wejściową.
- Nazwę pliku z macierzą wyjściową.

Na samym początku program (Manager Process) tworzy podaną (w drugim argumencie programu) liczbę procesów potomnych (Worker Process).



Dla każdego wpisu z pliku `lista` proces potomny mnoży odpowiednie wiersze macierzy A przez odpowiednie kolumny macierzy B, a wynik zapisuje w odpowiednie miejsce

macierzy *C* — każdy z procesów potomnych korzysta z całej macierzy *A*, a w przypadku macierzy *B*, tylko z określonych jej kolumn — patrz rysunek:



Przykładowo, dla dwóch procesów potomnych, pierwszy z nich mnoży odpowiednie wiersze macierzy *A* przez kolumny macierzy *B* oznaczone kolorem zielonym; drugi proces potomny, mnoży odpowiednie wiersze macierzy *A* przez kolumny macierzy *B* oznaczone kolorem niebieskim.
Czas działania procesu potomnego nie może przekroczyć podanej liczby sekund — trzeci argument programu. Po upłynięciu tego czasu każdy z procesów potomnych kończy swoje działanie, zwracając do procesu macierzystego poprzez kod wyjścia procesu liczbę wykonanych mnożeń — wymnożenie fragmentu macierzy *A* przez fragment macierzy *B* traktujemy jako jedno mnożenie. Proces, który zakończył mnożenie fragmentów, nie kończy działania — może być użyty do kolejnej operacji mnożenia. Program macierzysty pobiera statusy procesów potomnych, wypisuje raport: "Proces PID wykonał *n* mnożeń macierzy" i kończy swoje działanie. UWAGA! Nie używamy sygnałów, które są tematem następnych zajęć. Ponadto zakładamy, że jedynym dostępnym sposobem komunikacji procesów jest, w tym momencie, komunikacja poprzez plik — właściwe sposoby komunikacji procesów poznamy na dalszych ćwiczeniach.

Tworzenie pliku wynikowego może się odbywać na dwa sposoby — czwarty argument programu:

1. Procesy potomne zapisują wynik mnożenia w odpowiednie miejsce **wspólnego** pliku wynikowego — pamiętaj o odpowiednim użyciu blokady FLOCK.
2. Procesy potomne zapisują wynik mnożenia do **odrębnych** plików; osobny proces wywołuje jedną z funkcji rodziny `exec*()` w celu wykonania komendy `paste` — połączenie zawartości plików.

Napisz pomocniczy program, który:

- Tworzy określoną liczbę plików z treścią macierzy. Rozmiar tworzonych macierzy, dla odrębnych par macierzy, jest losowy $\in [Min, Max]$, gdzie: *Min*, *Max* są argumentami programu, przy czym należy zadbać, aby (dla danej pary) liczba kolumn macierzy *A* była równa liczbie wierszy macierzy *B*.

- Umożliwia sprawdzenie poprawności otrzymanych wyników mnożenia — implementacja własna lub przy wykorzystaniu dowolnej biblioteki do wykonywania testów jednostkowych

Zadanie 3. Zasoby procesów (35%)

Zmodyfikuj program z Zadania 2 tak, aby każdy proces (mnożący) miał nałożone pewne twarde ograniczenie na dostępny czas procesora oraz rozmiar pamięci wirtualnej. Wartości tych ograniczeń (odpowiednio w sekundach i megabajtach) powinny być przekazywane jako dwa ostatnie argumenty wywołania programu `macierz`. Ograniczenia powinny być nakładane przez proces potomny, w tym celu należy użyć funkcji `setrlimit()`. Zakładamy, że wartości nakładanych ograniczeń są dużo niższe (t.j. bardziej restrykcyjne) niż ograniczenia, które system operacyjny narzuca na użytkownika uruchamiającego `macierz`.

Zaimplementuj w `macierz` raportowanie zużycia zasobów systemowych dla każdego procesu potomnego, w tym czas użytkownika i czas systemowy. Realizując tę część zadania, zwróć uwagę na funkcję `getrusage()` i flagę `RUSAGE_CHILDREN`.