Programowanie równoległe. Przetwarzanie równoległe i rozproszone.
Laboratorium 2

Cel:

- przeprowadzenie pomiaru czasu CPU i zegarowego tworzenia procesów i wątków systemowych Linuxa
- nabycie umiejętności pisania programów wykorzystujących tworzenie wątków i procesów (wykład 1 – slajdy i nagranie <tworzenie wątków – minuta 35 i dalej>)

Zajęcia:

0. Przeczytanie uwag na temat pomiaru czasu wykonania programów na stronie przedmiotu (lab. 1 – zasady pomiaru czasu)
1. Utworzenie podkatalogu roboczego w katalogu *PR_lab* (np. *lab_2*).
2. Skopiowanie ze strony WWW do katalogu roboczego pliku „*fork_clone.tgz*”
3. Rozpakowanie plików, uruchomienie programów za pomocą *make* (dla osób, które nie zrealizowały części dodatkowej z *lab_1*, plik *libpomiar_czasu.tgz* zawiera zawartość katalogu *pomiar_czasu* (na tym samym poziomie jak *lab_1* i *lab_2*, jako podkatalogi *PR_lab*) – utworzenie biblioteki pomiaru czasu polega tylko na wykonaniu *make* w katalogu *pomiar_czasu*)
4. Uzupełnienie plików źródłowych o procedury pomiaru czasu i przeprowadzenie eksperymentu mierzącego czas tworzenia 1000 procesów i 1000 wątków – na jego podstawie obliczenie narzutu systemowego na tworzenie pojedynczego procesu i pojedynczego wątku:
   a) w wersji bez optymalizacji (-g -DDEBUG -O0)
   b) w wersji zoptymalizowanej (-O3) [*uwaga zmiana Makefile bez zmiany pliku źródłowego wymaga sekwencji: make clean; make w celu utworzenia plików binarnych dla nowych opcji*]

   (uwaga na optymalizacje kompilatora, który może usunąć operacje nie wywołujące żadnych efektów – standardowym sposobem uniknięcia optymalizacji jest wydruk na ekranie efektu działania wątków i procesów, w przypadku funkcji clone np. wartości zmiennej globalnej (jednak w taki sposób, żeby nie zaburzać znacząco czasu wykonania – czyli jednokrotny wydruk po długiej pętli, pętla musi być na tyle długa, żeby czas wydruku nie był porównywalny z czasem wykonania wszystkich iteracji pętli – można w tym celu zwiększyć liczbę iteracji, żeby czas wykonania pętli wyniósł co najmniej 0.1s); innym z możliwych testów jest sprawdzenie czasów realizacji dla wersji do debugowania i zoptymalizowanej i analiza różnic (wersja do debugowania nie powinna usuwać operacji z kodu, jeśli różnica czasu obu wersji sięga kilkudziesięciu lub kilkuset – może to wskazywać na usunięcie przez kompilator operacji z kodu źródłowego, które nie miały wpływu na ostateczny prezentowany użytkownikowi wynik).
   (można także zaprojektować i przeprowadzić odrębny eksperyment mierzący czas realizacji funkcji *exec*) **(ocena)**
   - *-> Jaka jest relatywna wielkość narzutu na tworzenie wątków, w porównaniu do narzutu na tworzenie procesów (jako narzut należy potraktować cały czas zegarowy wykonania funkcji fork lub clone)? Ile operacji arytmetycznych (szacunkowo) może wykonać procesor w czasie, który zajmuje tworzenie wątków? Ile operacji wejścia/wyjścia? Czy optymalizacja wpływa na czas tworzenia procesów i wątków? Co jest tego przyczyną?*
5. Napisanie nowego programu (najlepiej na podstawie *clone.c*, ale bez pętli tworzenia wątków), w którym bezpośrednio po sobie tworzone są dwa wątki do działania równoległego (*uwaga na synchronizację! - nie można wywoływać oczekiwania na koniec pracy jednego wątku przed uruchomieniem drugiego*). Wątki mogą wykonywać tę samą funkcję lub dwie różne funkcje (można wykorzystać funkcję *funkcja_watku* z *clone.c*). Wątki mają zwiększać w odpowiednio długiej pętli (co najmniej 100000 iteracji) wartości dwóch zmiennych: jednej globalnej (lub statycznej), drugiej lokalnej (na stosie – przekazanej jako argument). Obserwacja zachowania zmiennych poprzez wypisanie ich wartości: w funkcjach wątków po zakończeniu pętli, w funkcji main (oczywiście po zakończeniu pracy obu wątków!) **(ocena)**

- *-> Czy wartości zmiennych odpowiadają liczbie operacji wykonanych przez wątki? (należy obliczyć ile powinny wynosić wartości zmiennych po wykonaniu kodu i porównać z rzeczywiście otrzymanymi wartościami). Jak zachowują się zmienne globalne (statyczne), a jak zmienne lokalne w funkcjach wykonywanych przez wątki?*

---------- 3.0 ----------

Dalsze kroki dla podniesienia oceny:

1. Modyfikacja kodu programów, tak aby procedura wątku (w *clone.c*) i nowo utworzony proces (w *fork.c*) uruchamiały (za pomocą funkcji *exec*) program wypisujący na ekranie indywidualnie zaprojektowane dane (np. imię i nazwisko, numer procesu lub tp.)
2. W pierwszej wersji wystarczy odkomentować odpowiednie linie w plikach *fork.c* i *clone.c* oraz napisać i skompilować prosty program, nie pobierający argumentów, wypisujący dane na sztywno wpisane do kodu
    1. w wersji zaawansowanej można napisać program przyjmujący argumenty (*argc*, *argv*) i przesłać dane do wysłania jako argumenty programu, za pomocą funkcji *execv* (należy pamiętać, że ostatnim argumentem w tablicy wskaźników argv ma być NULL)
3. **(ocena)**

---------- 4.0 ----------

4. Analiza wywołania funkcji *clone()* (m.in. przeglądanie strony podręcznika „*man clone*”) i taka jej modyfikacja, aby jej działanie pokrywało się z działaniem funkcji *fork()*.
5. Analiza znaczenia parametrów funkcji *clone()* - wskaźnik stosu, opcje współdzielenia zasobów – testowanie wariantów opcji, np.:
    1. statyczna alokacja dużej tablicy w procedurze wątku i sprawdzenie jaki trzeba dobrać stos, aby procedura mogła zostać zrealizowana (standardowy rozmiar stosu w systemie można wyświetlić wywołując funkcję *limit (tcsh) lub ulimit (bash)*) **(ocena)**
        - *-> W jaki sposób można określić rozmiar stosu dla nowo tworzonego wątku? W jaki sposób można przekroczyć ten rozmiar? (jednocześnie testując jego wielkość) Ile wynosi domyślny rozmiar stosu?*
    2. próba otwarcia pliku w przypadku braku współdzielenia systemu plików **(ocena)**
    3. itd., itp.
6. Dowolne zadania polecane przez prowadzących

---------- 5.0 ----------

Warunki zaliczenia:

1. Obecność na zajęciach i wykonanie co najmniej kroków 1-6
2. Oddanie sprawozdania o treści i formie zgodnej z regulaminem ćwiczeń laboratoryjnych (z krótkim opisem zadania (cel, zrealizowane kroki, wnioski) oraz kodem źródłowym utworzonym przez siebie, umieszczonym przy opisie odpowiadających kroków). Sprawozdanie powinno zawierać konkretne wyniki pomiarów czasu wykonania konkretnych operacji (wklejone jako obrazy z zawartą informacją identyfikującą osobę przeprowadzającą obliczenia – patrz ogólne wskazówki tworzenia sprawozdań w regulaminie laboratoriów), a wnioski **porównanie wyników między sobą oraz porównanie z wynikami z poprzedniego laboratorium** – czasu wykonania pojedynczej operacji arytmetycznej i pojedynczej operacji wejścia/wyjścia, a także wnioski dotyczące zachowania zmiennych.
3. Symbol -> oznacza pytania, na które odpowiedzi ma dać laboratorium (odpowiedzi powinny znaleźć się w sprawozdaniu)