Imię, nazwisko

Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Sprawozdanie z laboratorium 1.

Celem laboratorium było nabycie (odświeżenie, pogłębienie) umiejętności realizacji pomiaru czasu wykonania fragmentów kodu oraz umiejętności posługiwania się narzędziem make do kompilacji i konsolidacji kodu.

W ramach zajęć zrealizowała(e)m następujące kroki:

- skopiowałam wskazane pliki (*Makefile, pomiar_czasu.c, pomiar_czasu.h, moj_program.c*) do katalogu roboczego lab_1
- funkcje pomiaru czasu z pliku pomiar_czasu.c korzystają z procedur systemowych: *getrusage(RUSAGE_SELF, &rp) i gettimeofday(&tp,&tzp).* Pierwsza z nich pobiera informacje o wykorzystaniu zasobów systemowych, między innymi czasu CPU, przez proces (*getresusage – get resource usage* – zwraca strukturę, której pola *ru_utime.tv_usec i ru_utime.tv_usec* pozwalają obliczyć zużycie czasu CPU przez proces w sekundach). Funkcja *gettimeofday* zwraca czas odmierzany od pewnego punktu, także w polach struktury będącej jej argumentem: *tv_sec i tv_usec*. Obie funkcje pozwalają uzyskać pomiar czasu z dokładnością ok. mikrosekundy, znacznie wyższą od dokładności funkcji *clock* języka C.
- funkcje i*nicjuj_czas() i drukuj_czas()* stanowią parę – pierwsza inicjuje zerami lokalną strukturę danych do odmierzania czasu, drugą drukuje różnicę aktualnych wartości pomiaru czasu i wartości zapisanych w strukturze. Funkcje *czas_zegara() i czas_CPU()* zwracają aktualne wartości odmierzanych czasów i wymagają co najmniej dwóch pomiarów w różnych punktach programu z zapisaniem wartości w zmiennych lokalnych do ustalenia czasu użytego pomiędzy punktami programu
- zmodyfikowałem plik źródłowy *moj_program.c* umieszczając dyrektywę dołączenia pliku nagłówkowego kodu pomiaru czasu oraz wywołania procedur pomiaru i wydruku czasu :

```
#include"pomiar_czasu.h"
....
inicjuj_czas();
for(i=0;i<liczba;i++){
    printf("%d ",k+i);
}
printf("\n");
drukuj_czas();
...
t1=czas_zegara(); t2=czas_CPU();
for(i=0;i<liczba;i++){
    a = 1.000001*a;
}
t1=czas_zegara()-t1; t2=czas_CPU()-t2;
printf("Czas wykonania %d operacji arytmetycznych: zegarowy %lf, CPU
%lf\n",liczba,t1,t2);
```

- zmodyfikowałem plik Makefile tak aby poprawnie tworzyć plik binarny *moj_program* poprzez:
  - dodanie plików *moj_program.c pomiar_czasu.h* w linii zależności pliku *moj_program.o*
  - wpisanie polecenia kompilacji z odpowiednimi opcjami pliku źródłowego *moj_program.c*

```
# jak uzyskac plik moj_program.o ?
moj_program.o: moj_program.c pomiar_czasu.h
	$(CCOMP) -c $(OPT) moj_program.c $(INC)
```

- utworzyłam i uruchomiłam kod
- przetestowałem działanie uzyskując następujące wyniki dla wersji do debugowania (dla każdej wartości przeprowadziłem pięć pomiarów i wybrałem najkrótszy):

```
49936 49937 49938 49939 49940 49941 49942 49943 49944 49945 49946 49947 49948 49949 49950 49951 4995
2 49953 49954 49955 49956 49957 49958 49959 49960 49961 49962 49963 49964 49965 49966 49967 49968 49
969 49970 49971 49972 49973 49974 49975 49976 49977 49978 49979 49980 49981 49982 49983 49984 49985
49986 49987 49988 49989 49990 49991 49992 49993 49994 49995 49996 49997 49998 49999
Czas wykonania 100000 operacji wejscia/wyjscia:
czas standardowy = 0.010000
czas CPU         = 0.018663
czas zegarowy    = 0.026584
Wynik operacji arytmetycznych: 1.210343
Czas wykonania 100000 operacji arytmetycznych: zegarowy 0.000864, CPU 0.000865
[1]-> ht
ESTERA
[1]-> pwd
/home/kbanas/PR_2020/lab/1
```

Otrzymane wyniki pozwalają obliczyć czas wykonania pojedynczej operacji:
- arytmetycznej (mnożenia):
  - czas CPU – $8{,}64*10^{-9}$
  - czas zegara – $8{,}64*10^{-9}$
- we/wy (wydruk pojedynczej liczby w terminalu z przesłaniem danych z serwera przez sieć):
  - czas CPU – $186{,}63*10^{-9}$
  - czas zegara – $265{,}84*10^{-9}$

(pomiar realizowany jest na serwerze, a wydruk jest lokalny – czas mierzony obejmuje obsługę wysłania przez sieć)

(to samo dla wersji zoptymalizowanej)

W ramach zadań dodatkowych zrealizowałam:
- przeniosłam pliki związane z pomiarem czasu do odrębnego katalogu pomiar_czasu
- w katalogu pomiar_czasu utworzyłam bibliotekę *libpomiar_czasu.a* z pliku *pomiar_czasu.o* poleceniem *ar -rs libpomiar_czasu.a pomiar_czasu.o*
- zmodyfikowałam plik *Makefile* tak aby zamiast z plików źródłowych i pośrednich pomiaru czasu korzystał z biblioteki:
  - usunąłem odniesienia do plików *pomiar_czasu.c i pomiar_czasu.o*
  - nadałem właściwe wartości symbolom LIB i INC – pierwszy wskazuje na położenie bibliotek (opcja -L) oraz dołączenie bibliotek przy konsolidacji (opcja -l), drugi (INC) wskazuje położenie plików nagłówkowych do włączania do plików z kodem źródłowym w trakcie kompilacji kodu
  - dodałem właściwe wykorzystanie symboli LIB i INC w trakcie kompilacji

```
# pliki naglowkowe
INC = -I./pomiar_czasu
# biblioteki
LIB = -L./pomiar_czasu -lpomiar_czasu -lm
# zaleznosci i komendy
moj_program: moj_program.o
	$(LINKER) $(OPT) moj_program.o -o moj_program $(LIB)
# jak uzyskac plik moj_program.o ?
moj_program.o: moj_program.c pomiar_czasu/pomiar_czasu.h
	$(CCOMP) -c $(OPT) moj_program.c $(INC)
```

- przetestowałem kompilacje kodu z nową wersja pliku Makefile

Wnioski:

- dzięki wykorzystaniu narzędzia make oraz odpowiednio modyfikowanego pliku Makefile udało się zrealizować sprawne i elastyczne tworzenie kodu binarnego z plików źródłowych
- pomiar czasu realizacji procedur wykazał, że:
  - czas realizacji procedur wejścia/wyjścia jest znacznie dłuższy niż czas realizacji operacji arytmetycznych
  - przy realizacji procedur wejścia/wyjścia czas wykorzystania procesora (czas CPU) jest niższy od czasu zegarowego (proporcja może zależeć od sposobu funkcjonowania środowiska graficznego systemu oraz od faktu czy pomiar realizowany jest dla operacji lokalnych czy wydruk odbywa się zdalnie)
- *wnioski dotyczące wersji do debugowania i zoptymalizowanej*