Marek, Kubicki

Programowanie równoległe.

Sprawozdanie z laboratorium 2.

1. Celem tych zajęć było wykonanie pomiaru czasów tworzenia wątków i procesów, porównanie narzutów tych operacji, oraz porównanie wyniku działań na zmiennych globalnych, oraz lokalnych.

Po pobraniu i rozpakowaniu materiałów potrzebnych do zajęć w odpowiednich folderach przystąpiłem do analizy i modyfikacji dostarczonych plików.

Poza plikami potrzebnymi do utworzenia biblioteki z końca poprzednich laboratoriów dostarczone zostały trzy pliki:
- Makefile – potrzebny do kompilacji pozostałych dwóch z plików z opcjami:

OPT = -O3 -Opcja służąca do optymalizacji
OPT = -g -DDEBUG -Opcja debugowa, zastępująca poprzednią kiedy chcemy wymusić brak optymalizacji programu

- clone.c – W tym pliku wykorzystana jest procedura clone() tworząca proces potomny na zarezerwowanym miejscu w stosie.

- fork.c – Podobnie jak w pliku clone, tutaj także tworzony jest proces potomny, jednakże tym razem procedurą fork().

Procedury fork() i clone() są do siebie dosyć podobne, fork() pozwala na łatwe stworzenie nowego procesu, clone() pozwala na większą kontrolę tego, jakie polecenia zostaną wykonane przez konkretne procesy.

W dalszej części laboratorium uzupełniłem wymienione wyżej pliki o procedury pomiaru czasu i wykonałem odpowiednie pomiary. Po odrzuceniu pomiarów znacząco błędnych otrzymałem wyniki załączone na dole dokumentu. Wynika z nich, że tworzenie procesów i wątków jest porównywalnie szybkie do operacji wejścia/wyjści, ale 10 wolniejsze niż operacje arytmetyczne w wersji debugowej i 100 razy wolniejsze niż operacje arytmetyczne w wersji zoptymalizowanej.

-> Wątki są szybsze niż procesy. Wątek to część procesu, oznacza to, że proces może mieć wiele wątków, które nie są od siebie odizolowane. Procesy są od siebie odizolowane, co oznacza, że nie dzielą one swojej pamięci pomiędzy sobą.

Porównując wyniki z laboratoriów 1 i 2 tworzenie wątków jest X razy szybsze niż proste operacje arytmetyczne. Oznacza to, że podczas tworzenia 1 wątku można wykonać około X operacji arytmetycznych.

Optymalizacja nie wpływa na samo tworzenie wątków, ale może ona przyśpieszyć operacje towarzyszące ich tworzeniu, lub wykonaniu. Może ona także pominąć tworzenie wątków w skrajnych przypadkach, przyśpieszając działanie programu.

2. W kolejnej części laboratorium dodałem tworzenie drugiego wątku w pliku clone.c, oraz zmodyfikowałem istniejącą funkcję w celu tak, aby przyjmowała wskaźnik do zmiennej lokalnej.
--> Po wypisaniu i porównaniu wartości zmiennych lokalnych i globalnych pomiędzy kilkoma wywołaniami kodu skompilowanego z optymalizacją i bez można zauważyć, że przy wykorzystaniu optymalizacji wartość zmiennej lokalnej wzrasta poprawnie, a przy jej braku wartość zmiennej różni się pomiędzy wywołaniami.

Załączniki:

Tabele ze zmierzonymi czasami operacji wykonanych w laboratoriach 1 i 2. Potrzebne do porównania długości operacji

Lab. 1:

10000	OP. In/out		OP. Arytmetyczne	
Wersje Debugowania	Czas zegara	Czas CPU	Czas zegara	Czas CPU
	1,484906	0,149768	0,49476	0,368468
	1,114468	0,122379	0,502477	0,330702
	1,591029	0,123172	0,513455	0,409313
Średnia:	1,396801	0,131773	0,503564	0,369494333
Czas 1 operacji	1,40E-04	1,32E-05	5,04E-05	3,69E-05

10000	OP. In/out		OP. Arytmetyczne	
Wersje do optymalizacji	Czas zegara	Czas CPU	Czas zegara	Czas CPU
	1,468254	0,144623	0,0050417	0,00378856
	1,191308	0,149213	0,00480317	0,00480317
	1,324293	0,152852	0,00471157	0,00206583
Średnia:	1,327951667	0,148896	0,004852147	0,00355252
Czas 1 operacji	1,33E-04	1,49E-05	4,85E-07	3,55E-07

Lab. 2:

1000	clone		fork	
Wersje Debugowania	Czas zegara	Czas CPU	Czas zegara	Czas CPU
	0,145263	0,005876	0,372126	0,001033
	0,127555	0,006485	0,371628	0,008711
	0,132134	0,007596	0,325689	0,001434
Średnia:	0,134984	0,006652333	0,356481	0,003726
Czas 1 operacji	1,35E-04	6,65E-06	3,56E-04	3,73E-06

1000	clone		fork	
Wersje do optymalizacji	Czas zegara	Czas CPU	Czas zegara	Czas CPU
	0,122024	0,004162	0,168247	0,005469
	0,125381	0,002849	0,180189	0,006128
	0,168247	0,005469	0,165843	0,006834
Średnia:	0,138550667	0,00416	0,171426333	0,006143667
Czas 1 operacji	1,39E-04	4,16E-06	1,71E-04	6,14E-06

Ważne fragmenty kodu:

Wywoływanie funkcji clone:

```
int mai = 0;
pid = clone( &funkcja_watku, (void *) stos+ROZMIAR_STOSU,
       CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, &mai );

pid2 = clone( &funkcja_watku, (void *) stos2+ROZMIAR_STOSU,
       CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, &mai );

waitpid(pid, NULL, __WCLONE);
waitpid(pid2, NULL, __WCLONE);
```

Wywoływana funkcja:

```
int funkcja_watku( void* arg )
{
int i;
int zmienna_lokalna = 0;
for(i=0; i<100000;i++)
{
  zmienna_lokalna++;
  zmienna_globalna++;
}
   *(int*)arg += zmienna_lokalna;
printf("Clone nr: %d\n",zmienna_globalna);
printf("zmienna ll: %d    clone %d\n", zmienna_lokalna,zmienna_globalna2);
printf("zmienna gl: %d    clone %d\n",zmienna_globalna,zmienna_globalna2);
zmienna_globalna2++;
  return 0;
}
```

Efekt uruchomienia programu, kolizja różnych wątków, różne wartości zmiennych: