Monika Krakowska

Programowanie równolegle

Sprawozdanie z laboratorium nr 2

Cel laboratorium:

Celem laboratorium było przeprowadzenie pomiary czasu CPU i zegarowego tworzenia procesów i wątków systemowych Linuxa oraz nabycie umiejętności pisania programów wykorzystujących tworzenie wątków i procesów.

Przebieg ćwiczenia:

1) Przygotowanie środowiska roboczego:

-Utworzyłam podkatalog roboczy o nazwie lab_2 poleceniem mkdir w utworzonym na pierwszych zajęciach katalogu PR_lab.

-Ze strony przedmiotu pobrałam plik fork_clone.tgz i wypakowałam go do katalogu roboczego

-Za pomocą komend make oraz make clone skompilowałam oba programy po czym uruchomiłam je (./fork, ./clone)

2) Uzupełnienie plików źródłowych:

-Uzupełniłam oba pliki źródłowe o procedury pomiaru czasu (inicjuj_czas() oraz drukuj_czas()), korzystając z plików związanych z pomiarem czasu przygotowanych na laboratorium nr 1

3) Eksperyment:

-Przeprowadziłam eksperyment mierzącego czas tworzenia 1000 procesów i takiej samej ilości wątków uzyskując następujące wyniki:

4) Wyniki pomiarów:

-1000 procesów:

tworzenie 1000 procesow		czas standard	czas CPU	czas zegarowy
	pomiar 1	0,151537	0,019646	0,478329
	pomiar 2	0,173481	0,005596	0,488841
	pomiar 3	0,161283	0,008716	0,470955
	pomiar 4	0,147292	0,012203	0,489675
	średnia:	0,158398250	0,011540250	0,481950000
tworzenie 1 procesu				
	pomiar 1	0,000151537	0,000019646	0,000478329
	pomiar 2	0,000173481	0,000005596	0,000488841
	pomiar 3	0,000161283	0,000008716	0,000470955
	pomiar 4	0,000147292	0,000012203	0,000489675
	średnia:	0,000158398	0,000011540	0,000481950

-1000 wątków:

tworzenie 1000 watkow				
	pomiar 1	0,04731	0,012686	0,112419
	pomiar 2	0,029086	0,0104	0,095549
	pomiar 3	0,032648	0,003628	0,086882
	pomiar 4	0,031917	0,008479	0,098571
	średnia:	0,035240250	0,008798250	0,098355250
tworzenie 1 watku				
	pomiar 1	0,00004731	0,000012686	0,000112419
	pomiar 2	0,000029086	0,0000104	0,000095549
	pomiar 3	0,000032648	0,000003628	0,000086882
	pomiar 4	0,000031917	0,000008479	0,000098571
	średnia:	0,000035240	0,000008798	0,000098355

po sprawdzeniu dla wersji do debugowania i zoptymalizowanej-
roznice nie sa znaczace, oznacza to ze kompilator nie usunal zadnych operacji z kodu zrodlowego, pomijam wiec te wyniki

5) Nowy program- clone2

-Na podstawie programu clone.c napisałam zbliżony program- clone2.c w którym, bezpośrednio po sobie tworzone są dwa wątki do działania równoległego. Wątki zwiększają w pętli dwie zmienne- zmienną globalna i zmienną lokalną.

-Zmodyfikowałam plik Makefile aby analogicznie do clone, kompilował również program clone2

-Uruchomiłam program clone2

-Uzyskane wyniku znajdują się w załączniku na końcu sprawozdania

6) Obliczenia:

-Aby obliczyć ile operacji arytmetycznych (szacunkowo) może wykonać procesor w czasie, który zajmuje tworzenie wątków i ile operacji wejścia/wyjścia, musimy cofnąć się do wyników uzyskanych w laboratorium nr 1 (Załącznik 5.). Po przeprowadzeniu obliczeń uzyskujemy następujące wnioski: procesor może wykonać około 543703 operacji arytmetycznych w czasie, który zajmuje mu tworzenie 1000 wątków oraz około 8940 operacji wejścia/wyjścia.

Wnioski:

Na podstawie wyników możemy ocenić efektywność tworzenia procesów i wątków. Przy tworzeniu procesów mamy znacznie większy narzut niż przy tworzeniu wątków.

Porównując tworzenie wątków jest bardziej efektywne pod kątem zasobów CPU. Wątki mają mniejszy narzut na zasoby systemowe, co przejawia się krótszym czasem ich tworzenia.

W programie clone2, gdzie działały równolegle dwa wątki wyniki pokazały, że oba mogą niezależnie modyfikować swoje zmienne lokalne, ale operacje na zmiennych globalnych wymagają synchronizacji, w celu uniknięcia konfliktów.

W moim przypadku zarówno zmienne lokalne jak i zmienna globalna końcowy miały wynik, którego się spodziewaliśmy, zatem można wysnuć wniosek, że nie doszło do „wyścigu" wątków.

Laboratorium nauczyło mnie zrozumieć różnice miedzy procesami a wątkami, dokonywać pomiarów wydajności oraz pisania prostych programów wielowątkowych.

Załącznik 1. Kod programu clone.c:

```
#define _GNU_SOURCE

#include<stdlib.h>

#include<stdio.h>

#include<unistd.h>


#include <sys/types.h>

#include <sys/wait.h>

#include <sched.h>

#include <linux/sched.h>


#include "pomiar_czasu.h"


int zmienna_globalna=0;


#define ROZMIAR_STOSU 1024*64


int funkcja_watku( void* argument ){
```

```
 //zmienna_globalna++;
 return 0;
}
int main(){
 void *stos;
 pid_t pid;
 int i;
 stos = malloc( ROZMIAR_STOSU );
 if (stos == 0) {
  printf("Proces nadrzędny - blad alokacji stosu\n");
  exit( 1 );
 }
inicjuj_czas();
 for(i=0;i<1000;i++){
  pid = clone( &funkcja_watku, (void *) stos+ROZMIAR_STOSU,
                        CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, 0 );
  waitpid(pid, NULL, __WCLONE);
 }

printf("Czas tworzenia 1000 watkow:\n");
 drukuj_czas();
  int wynik;
  wynik=execv("./program",NULL);
  if(wynik==-1)
   printf("Proces potomny nie wykonal programu\n");
 free( stos );
}
```

Załącznik 2. Kod programu clone2.c:

```
#define _GNU_SOURCE
#include<stdlib.h>
```

```
#include<stdio.h>
#include<unistd.h>

#include <sys/types.h>
#include <sys/wait.h>
#include <sched.h>
#include <linux/sched.h>

#include "pomiar_czasu.h"

int zmienna_globalna=0;
#define ROZMIAR_STOSU 1024*64

int funkcja_watku( void* argument ){
int zmienna_lokalna = *(int*) argument;

for(int i=0; i<1000000; i++){
 zmienna_globalna++;
 zmienna_lokalna++;
 }
 printf("Wartosc zmiennej lokalnej= %d, wartosc zmiennej globalnej= %d\n", zmienna_lokalna,
zmienna_globalna);

 /* int wynik; */
 /* wynik=execv("./program",NULL); */
 /* if(wynik==-1) */
 /*   printf("Proces potomny nie wykonal programu\n"); */

 return 0;
}
```

```
int main(){
  void *stos;
  void *stos2;
  pid_t pid;
  pid_t pid2;
  int i;

  stos = malloc( ROZMIAR_STOSU );
  stos2 = malloc( ROZMIAR_STOSU );

  if (stos == 0) {
    printf("Proces nadrzędny - blad alokacji stosu\n");
    exit( 1 );
  }
  if (stos2 == 0) {
    printf("Proces nadrzędny - blad alokacji stosu\n");
    exit( 1 );
  }
 inicjuj_czas();
    pid = clone( &funkcja_watku, (void *) stos+ROZMIAR_STOSU,
                        CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, &i );
    pid2 = clone( &funkcja_watku, (void *) stos2+ROZMIAR_STOSU,
                        CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, &i );
    waitpid(pid, NULL, __WCLONE);
    waitpid(pid2, NULL, __WCLONE);
// printf("Czas tworzenia 1000 watkow:\n");
//  drukuj_czas();
//  free( stos );
//  free( stos2 );
 printf("Wartosc zmiennej globalnej= %d\n", zmienna_globalna);}
```

Załącznik 3. Kod programu fork.c:

```
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include"pomiar_czasu.h"

int zmienna_globalna=0;

int main(){
 int pid, wynik, i;
 inicjuj_czas();
  for(i=0;i<1000;i++){
     pid = fork();

  if(pid==0){
        zmienna_globalna++;
    //char arg1[] = "/bin/ls";
    //char arg2[] = ".";
    //char* arg[] = {arg1,arg2,NULL};
    //char* arg[] = {"/bin/ls",".",NULL};
   // wynik=execv("/bin/ls",arg);
     exit(0);
      } else {
       wait(NULL);
      }
   }
 drukuj_czas();
}
```

Załącznik 4. Kod Makefile:

```
# kompilator c
CCOMP = gcc

# konsolidator
LOADER = gcc

# opcje optymalizacji:
# wersja do debugowania
#OPT = -g -DDEBUG
# wersja zoptymalizowana do mierzenia czasu
 OPT = -O3

# pliki naglowkowe
INC = -I../pomiar_czasu

# biblioteki
LIB = -L../pomiar_czasu -lpomiar_czasu -lm

# zaleznosci i komendy
fork: fork.o
	$(LOADER) $(OPT) fork.o -o fork $(LIB)

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

# zaleznosci i komendy
clone: clone.o
	$(LOADER) $(OPT) clone.o -o clone $(LIB)
```

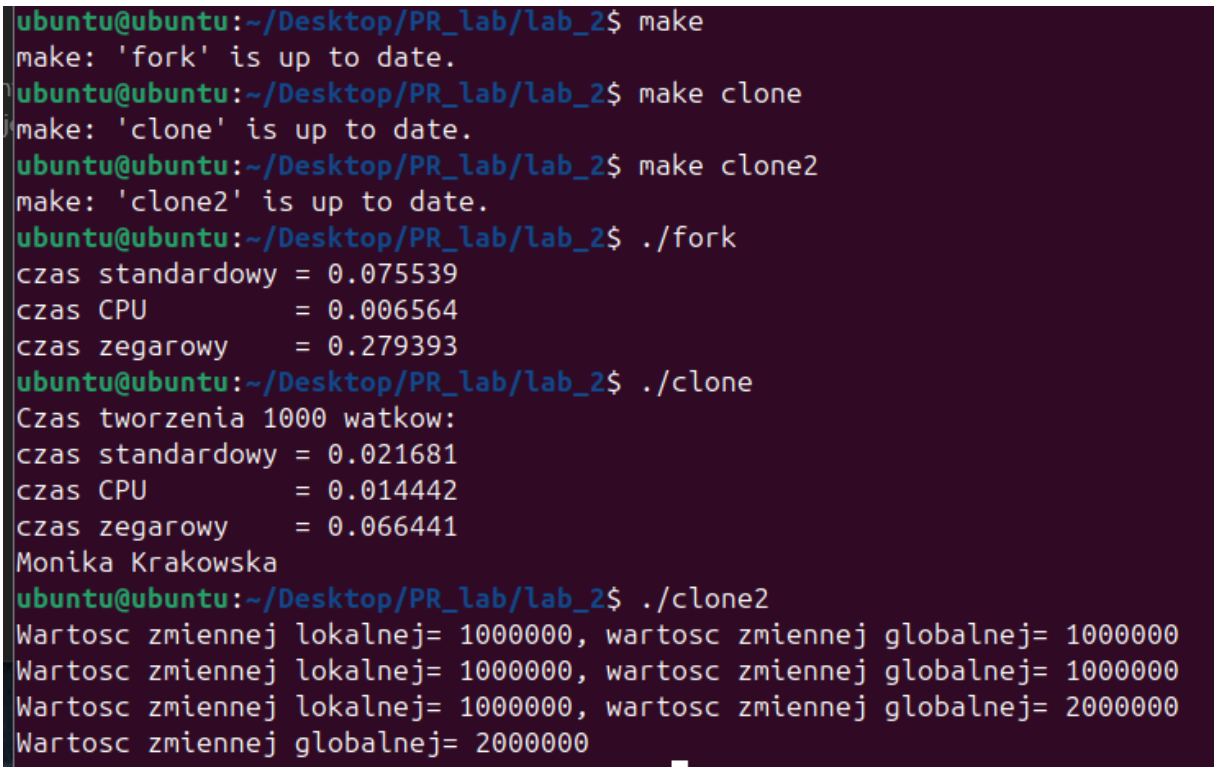
```
clone2: clone2.o
	$(LOADER) $(OPT) clone2.o -o clone2 $(LIB)

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

clone2.o: clone2.c ../pomiar_czasu/pomiar_czasu.h
	$(CCOMP) -c $(OPT) clone2.c $(INC)

clean:
	rm -f *.o fork clone clone2
```

Załącznik 4. Niektóre wydruki z terminala:



Załącznik 5. Przykładowe wyniki uzyskane przeze mnie w laboratorium nr 1:

```
Czas wykonania 100000 operacji wejscia/wyjscia:
czas standardowy = 0.011005
czas CPU         = 0.011005
czas zegarowy    = 0.045076
Wynik operacji arytmetycznych: 1.210343
Czas wykonania 100000 operacji arytmetycznych:
Czas zegara: 0.000181, czas CPU: 0.000182
```

c