9.10.2023	Łukasz Oprych Gr.5 ITE	Temat: Procesy, Wątki, Fork/Clone

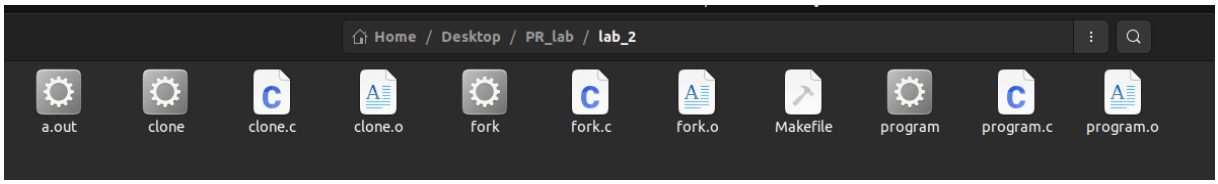
Cel:

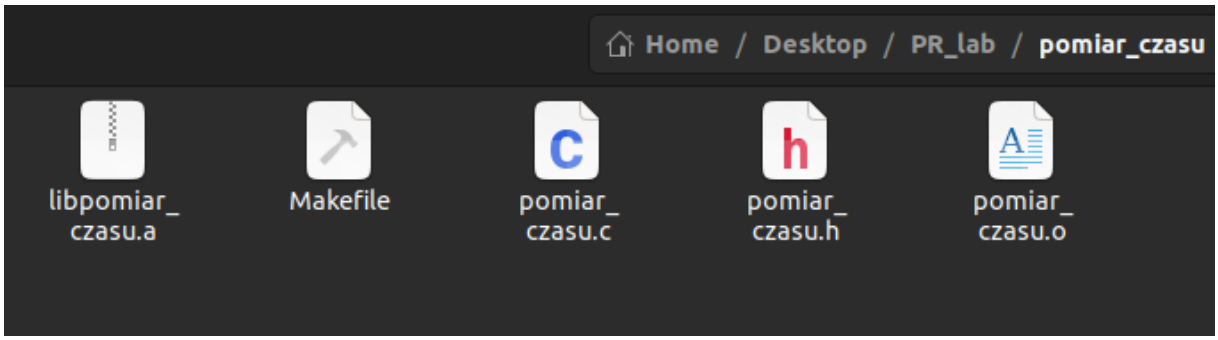
Zapoznanie się z procesami i wątkami w systemie operacyjnym Linux z wykorzystaniem funkcji fork() i clone().

Wykonanie ćwiczenia:

1. Utworzenie katalogu lab_2 oraz skopiowanie do niego pliku fork_clone.tgz z kodem źródłowym, które rozpakowano w katalogu oraz przygotowanie katalogu pomiar_czasu z kodem z zajęć poprzednich, który wykorzystano do pomiarów czasu.

   Struktura katalogowa:





2. Uzupełnienie plików źródłowych o procedury pomiaru czasu
   Wywołanie w clone.c oraz fork.c funkcji inicjuj_czas() przed pętla, oraz drukuj_czas() po wykonaniu pętli.

   Program fork.c:

```c
int main(){

  int pid, wynik, i;
  inicjuj_czas();
  for(i=0;i<1000;i++){
    pid = fork();
    if(pid==0){
      zmienna_globalna++;
      exit(0);
    } else {
      wait(NULL);
    }
  }
  drukuj_czas();

}
```

3. Następnie skompilowano oba programy za pomocą wywołania komendy make fork i make clone. Następnie wykonano 3 pomiary czasowe:

   Dla wersji niezoptymalizowanej:

Wersja DDEBUG	Czas tworzenia wątku (clone)		Czas tworzenia procesu (fork)	
Niezoptymalizowana	CPU (s)	Czas Clock (s)	CPU (s)	Clock (s)
Pomiar 1	0,00105	0,575029	0,002158	0,319457
Pomiar 2	0,001257	0,578611	0,00218	0,410047
Pomiar 3	0,001381	0,584547	0,003073	0,432571
Uśredniony wynik	0,001229333	0,579395667	0,002470333	0,387358333

   Oraz wersji zoptymalizowanej:

Wersja OPT -o3	Czas tworzenia wątku (clone)		Czas tworzenia procesu (fork)	
Zoptymalizowana	Czas CPU (s)	Czas Clock (s)	CPU (s)	Clock (s)
Pomiar 1	0,002273	0,532287	0,003066	0,408888
Pomiar 2	0,002306	0,549074	0,002205	0,378984
Pomiar 3	0,00218	0,523954	0,002996	0,488509
Uśredniony wynik	0,002253	0,535105	0,002755667	0,425460333

4. Utworzono nowy plik program.c, gdzie zainicjowano dwa wątki, które inkrementowały 100000 razy dwie zmienne, globalną oraz lokalną i czekały na zakończenie swojego działania. Wartości zmiennych były przesyłane za pomocą wskaźnika do wątku. Na koniec mogliśmy również sprawdzić jakie finalnie były wartości zmiennej lokalnej i globalnej.

   Wykonywana funkcja wątku:

```c
int funkcja_watku( void* argument )
{
  int local_value = *(int *)argument;
    for(int i = 0; i < 1e+5; i++){
        zmienna_globalna++;
        local_value++;
    }
    printf("Wewnatrz funkcji:\n");
    printf("Zmienna globalna: %d\n",zmienna_globalna);
    printf("Zmienna lokalna: %d\n",local_value);

  return 0;
}
```

Zainicjowanie stosów, wątków, funkcja oczekiwania wątków:

```c
int main()
{

  void *stos, *stos1;
  pid_t pid, pid1;
  int i;
  int local_value = 0;

  stos = malloc( ROZMIAR_STOSU );
  stos1 = malloc( ROZMIAR_STOSU );
  if (stos == 0) {
    printf("Proces nadrzędny - blad alokacji stosu\n");
    exit( 1 );
  }

    pid = clone( &funkcja_watku, (void *) stos+ROZMIAR_STOSU,CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, &local_value );
    pid1 = clone( &funkcja_watku, (void *) stos1+ROZMIAR_STOSU, CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, &local_value );

    waitpid(pid, NULL, __WCLONE);
    waitpid(pid1, NULL, __WCLONE);

  printf("Wewnatrz maina:\n");
  printf("Zmienna globalna: %d\n",zmienna_globalna);
  printf("Zmienna lokalna: %d\n",local_value);
  free( stos );
  free( stos1 );
}
```

Wynik wykonania program.c

Wersja niezoptymalizowana:

```
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_2$ ./program
Wewnatrz funkcji:
Zmienna globalna: 142644
Zmienna lokalna: 100000
Wewnatrz funkcji:
Zmienna globalna: 190867
Zmienna lokalna: 100000
Wewnatrz maina:
Zmienna globalna: 190867
Zmienna lokalna: 0
```

Wersja zoptymalizowana:

```
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_2$ ./program
Wewnatrz funkcji:
Zmienna globalna: 100000
Zmienna lokalna: 100000
Wewnatrz funkcji:
Zmienna globalna: 200000
Zmienna lokalna: 100000
Wewnatrz maina:
Zmienna globalna: 200000
Zmienna lokalna: 0
```

Wnioski:

- Aby uzyskać równolegle działający program, możemy stworzyć procesy za pomocą funkcji fork(), oraz dodatkowe wątki za pomocą funkcji clone(). Tworzenie wątku zajmuje mniej czasu, niż tworzenie osobnego procesu zgodnie z uzyskanymi wynikami.

- Z wyników prowadzonych pomiarów, można zauważyć, że tworzenie procesów oraz wątków w programie w wersji zoptymalizowanej ma dłuższe czasy CPU niż w wersji niezoptymalizowanej, w teorii optymalizacja powinna skrócić czas wykonywania operacji. Nieoczekiwane wyniki mogły być z winy sprzętu bądź oprogramowania.

- W sporządzonych wynikach tworzenia dwóch wątków działających równolegle w program.c zmienna lokalna na koniec ma wartość, która jest oczekiwana po wykonaniu 100000 iteracji. Zmienna globalna za to w przypadku kodu niezoptymalizowanego ma wynik różny od oczekiwanego ze względu na współdzielenie pamięci przez oba wątki, powoduje to, że gdy pierwszy wątek pobiera wartość zmiennej gdy została zmieniona przez drugi wątek. W przypadku kompilowania programu używając optymalizacji zmienna globalna, osiąga już oczekiwane wartości.