MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM Va

THREAD DI LINUX

A. T U J U A N

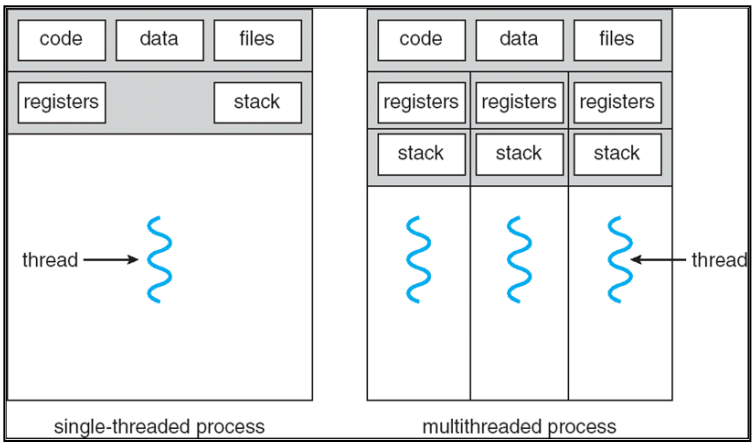
Memahami thread di Linux.

B. DASAR TEORI

1. Proses dan Thread

Tiap thread memiliki karakteristik sebagai berikut:

- Sebuah threadID
- Sebuah program counter
- Sebuah set register
- Sebuah stack



Gambar 1 Perbandingan Single Thread dan Multi Thread

Thread menggunakan sumber daya bersama sebagai berikut:

- Bagian kode
- Bagian data
- Sumber daya sistem operasi (contoh: opened file, signals, etc)

Karena threads berbagi memori dan data yang sama, maka kewaspadaan harus benar-benar diperhatikan ketika menulis kode multithread karena satu thread bisa jadi menimpa data threads yang lain. Thread memiliki kelebihan-kelebihan dibanding pembuatan proses baru, terutama karena overhead yang jauh lebih kecil ketika membuat dan menghancurkan thread baru dibandingkan proses. Selain itu, komunikasi antar thread sangat cepat dibandingkan komunikasi antar proses.

C. REFERENSI

Jonathan Macey. **Linux Systems Programming**. 2005.

C. LANGKAH – LANGKAH

Percobaan 1 : Thread

1. Membuat Thread sederhana. Beri nama `thread1.c`

```
#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
// we must make the compiler aware that this program
// is to use threads so the thread safe libraries must be used.
// To do this we define the _REENTERANT flag
#define _REENTRANT

//next we create a simple function for out thread
void *ThreadRoutine(int number)
{
    while(1) // loop forever
    {
        printf("pid %d : thread %d running\n",getpid(), number);
        sleep(number); // sleep for a time passed as a parameter
    }
}
int main(void)
{
    int t;

    pthread_t tid[5]; // an array to keep track of the threads
    // now loop through and create 4 threads passing t as the parameter
    for (t=1; t<5; t++)
            pthread_create(&tid[t],NULL,(void *)ThreadRoutine, &t);

    // now the parent loops and sleeps for 10
    while(1)
            {
             printf("pid %d : parent running\n", getpid());
             sleep(10);
            }
    exit(1);
}
```

Compile program dengan sintaks sebagai berikut:

```
gcc -Wall -g thread1.c -o thread1 -l pthread
```

Eksekusi dengan perintah: `./thread1`

Amati dan catat hasil yang ditampilkan.

2. Menerapkan `pthread_join()`. Beri nama `threadJoin.c`

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

#define _REENTRANT

//next we create a simple function for out thread
void *ThreadRoutine(int number)
{
    printf("thread %d running\n",number);
    sleep(number); // sleep for a time passed as a parameter
    printf("thread %d finish, joining… \n",number);
}
int main(void)
{
    int t;
    pthread_t tid; // an array to keep track of the threads
    // now loop through and create 4 threads passing t as the parameter
    for (t=1; t<5; t++)
        pthread_create(&tid,NULL,(void *)ThreadRoutine,(int *)t);

    // now the calling process waits for the thread to finish
    pthread_join(tid,NULL);
    printf("All kids joined, parent running\n");
    exit(1);
}
```

Compile program dengan sintaks sebagai berikut:

```
gcc -Wall -g threadJoin.c -o threadJoin -l pthread
```

Eksekusi dengan perintah: `./threadJoin`

Amati dan catat hasil yang ditampilkan.

3. Menerapkan `pthread_detach()`. Beri nama `threadDetach.c`

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

#define _REENTRANT
//next we create a simple function for out thread
void *ThreadRoutine(int number)
{
    int i;
    for (i=0; i<10; i++) //loop to give the thread something to do
        {
             printf("thread %d running %d\n",number,i);
             sleep(number); // sleep for a time passed as a parameter
        }
}

int main(void)
{
    pthread_t tid1,tid2; // create 2 thread id's
    //now create two threads
    pthread_create(&tid1,NULL,(void *)ThreadRoutine,(int *)1);
    pthread_create(&tid2,NULL,(void *)ThreadRoutine,(int *)2);
    pthread_detach(tid1); //we will now detach thread 1

    if(pthread_join(tid1,NULL)>0) // now try to join it
            printf("unable to join thread 1\n");
    if(pthread_join(tid2,NULL)>0) // and now join thread 2
            printf("unable to join thread 2\n");

    printf("parent finished\n");
    exit(1);
}
```

Compile program dengan sintaks sebagai berikut:

```
gcc -Wall -g threadDetach.c -o threadDetach -l pthread
```

Eksekusi dengan perintah: `./threadDetach`

Amati dan catat hasil yang ditampilkan.

Ulangi percobaan ini dengan mengubah bagian `pthread_detach(tid1)` menjadi `pthread_detach(tid2)` pada kode threadDetach.c

Amati dan catat hasil yang ditampilkan.

4. Menerapkan variabel global. Beri nama `globaldata.c`

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

int glob_data = 5 ;

void *kidfunc(void *p)
{
      printf ("Kid here. Global data was %d.\n", glob_data) ;
      glob_data = 15 ;
      printf ("Kid Again. Global data was now %d.\n", glob_data) ;
}

int main ( )
{
      pthread_t kid ;
      pthread_create (&kid, NULL, kidfunc, NULL) ;
      printf ("Parent here. Global data = %d\n", glob_data) ;
      glob_data = 10 ; pthread_join (kid, NULL) ;
      printf ("End of program. Global data = %d\n", glob_data) ;
}
```

Compile program dengan sintaks sebagai berikut:

```
gcc -Wall -g globaldata.c –o globaldata –l pthread
```

Eksekusi dengan perintah: `./globaldata`

Amati dan catat hasil yang ditampilkan.

D. EVALUASI

1. Jelaskan pengaruh nilai `sleep` pada program **thread1.c**
2. Jelaskan tujuan penggunaan `pthread_join` pada program **threadJoin.c**
3. Jelaskan fungsi `pthread_detach` pada program **threadDetach.c**
4. Pada percobaan **globaldata.c** apakah suatu variabel global dapat diakses oleh semua thread? Apa buktinya?