






CSE312-HW2 REPORT

 Created	@May 28, 2021
 Number	171044036
 Operating Systems	HomeWork Lecture Report
 Property	 Baran Hasan Bozduman

Definition

In this homework the thing asked for from us implementing Thread functionalities in spim as new syscalls.

And according to timer interrupt given in source files in every 100ms there will be an interrupt and we will give the cpu resources to other threads according to Round Robin scheduling.

Round Robin Scheduling Algorithm

The algorithm put the processes/threads in queue and execute them in order in this case it takes 100ms to change other thread.

Firstly we will have an init process which is the main thread or main process in the program and it will implement merge sort algorithm.

Secondly we will implement producer consumer program by implementing it using threads first it will use mutexes and in the second one they won't use it.

Design

I think as it will have a main thread which is depends on an init process. In the init process, also we will have different threads one of them will be used for the merge sort and other operations. also we should have a thread table to keep all or threads and one of thread will be a struct data structure.

Implementation

Thread Table

```

struct ThreadTABLE
{
    pid_t thread_id;
    char thread_name[50];
    int program_counter;
    char thread_state[20];
    mem_word *thread_stack_seg;
    short *thread_stack_seg_h;
    BYTE_TYPE *thread_stack_seg_b;
    mem_addr thread_stack_bot;
};

```

As above we will have some data structures all threads should have

thread id: Each thread should have an id to distinct from each other.

thread name: thread name can be used to make easy of users operations

program counter: Program counter will keep the last place that thread executed

thread state: thread state will keep run and ready states we don't need block states because we dont need to use them according to our structure run nd ready states provides enough.

stack areas: Stack is used to keep the variables data of the specified thread

For each thread you should determine the stack area and in my design user should allocate the stack and stores it in the \$a1 register and i determine the stack addresses according to that input as below and also i strode the thread address in \$a0 so when the threads turn is up it can know where to start.

```

la $a0, t2
addi $sp, $sp, -1024
la $a1, ($sp)
addi $sp, $sp, 1024
li $v0, 18
syscall

```

Thread Creating

As you see below i create a new thread and give them program counter and stack addresses, which is got from above. the example below its arranged for 3 thread but it can be increased and we can give the names dynamically.

```
case THREAD_CREATE:
{
    char temp[3][20]={{"main"}, {"t1"}, {"t2"}};
    int i;
    for ( i = 0; i < 3; i++)
    {
        if(threads[i]==NULL){
            threads[i]= new ThreadTABLE();
            break;
        }
    }
    printf("enter %d, %s, %d, %d, \n",i,temp[i] ,R[REG_A0],R[REG_A1]);
    threads[i]->thread_id=i;
    strcpy(threads[i]->thread_name,temp[i]);
    threads[i]->program_counter= R[REG_A0];threads[i]->thread_stack=
    threads[i]->thread_stack_seg=stack_seg;
    threads[i]->thread_stack_seg_h=stack_seg_h;
    threads[i]->thread_stack_seg_b=stack_seg_b;
    strcpy(threads[i]->thread_state, "READY");
    printf("exit %d, %s, %d, %d, \n",threads[i]->thread_id,i,threads[i]->thread_stack,
    threads[i]->thread_stack_seg_h);

    break;
}
```

Thread Join

When it comes to thread join i tried to run same thread again and again until the thread has done with its job it gives the priority other thread until all threads done with its job then it returns the main and finish the rest of program.

```

case THREAD_JOIN:
{
    int flag =0;
    int i=0;
    for(int j=0; j<3; j++)
        for ( i = 0; i < 3; i++)
        {
            if (threads[i]->thread_state != 'D')
            {
                flag=1;
                break;
            }
        }
    if(flag == 1){
        while(PC != threads[i]->program_counter){
            PC=threads[i]->program_counter;
            stack_bot= threads[i]->thread_stack_bot;
            stack_seg= threads[i]->thread_stack_seg;
            stack_seg_h=threads[i]->thread_stack_seg_h;
            stack_seg_b=threads[i]->thread_stack_seg_b;
        }
        threads[i]->thread_state = 'D';
        flag =0;
    }
}

```

Thread Exit

Before it calls thread exit it puts thread id into the \$a0 register then it makes a syscall after that in my algorithm i find that thread and convert it states to 'D' which means done so whenever there will be timer interrupt. The algorithm see the thread is done so it wont reschedule it again.

```

case THREAD_EXIT:
{
    int t_id_exit=R[REG_A0];
    int i;
    for ( i = 0; i < 3; i++)
    {
        if (t_id_exit == threads[i]->thread_id)
        {
            break;
        }
    }

    if(threads[i]->thread_state != 'D'){
        threads[i]->thread_state = 'D';
    }

    break;
}

```

Timer Handler

In each 100ms We get a timer interrupt and we should handle that and we should get the next thread in line as you see above, firstly i store the all elements in the related thread table and then i execute an algorithm which performs the round robin scheduling. To obtain that i used an array as a circular array.

```

    threads[glob_counter]->program_counter = PC;
    threads[glob_counter]->thread_stack_bot=stack_bot;
    threads[glob_counter]->thread_stack_seg=stack_seg;
    threads[glob_counter]->thread_stack_seg_h=stack_seg_h;
    threads[glob_counter]->thread_stack_seg_b=stack_seg_b;
    glob_counter+=1;
    if (glob_counter ==3)
    {
        glob_counter=0;
    }

    int i;
    for ( i = glob_counter; i < 3; i++)
    {
        if(strcmp(threads[i]->thread_state, "DONE")){
            //t_thread= threads[i];
            break;
        }
        if(i==2){
            i=0;
        }
    }
    //printf("HERE\n");
    glob_counter = i;
    PC=threads[i]->program_counter;
    stack_bot= threads[i]->thread_stack_bot;
    stack_seg= threads[i]->thread_stack_seg;
    stack_seg_h=threads[i]->thread_stack_seg_h;
    stack_seg_b=threads[i]->thread_stack_seg_b;

```

Output

As I researched the approach above should be right but there is a problem with output it doesn't switch second time and i really spend lots of time for that homework problem so i could not implement rest of it.

