CSE-312

OPERATING SYSTEMS

SPRING 2024

ONUR ATASEVER

210104004087

In this section, I added extra priority to the tasks and scheduler in Part A.

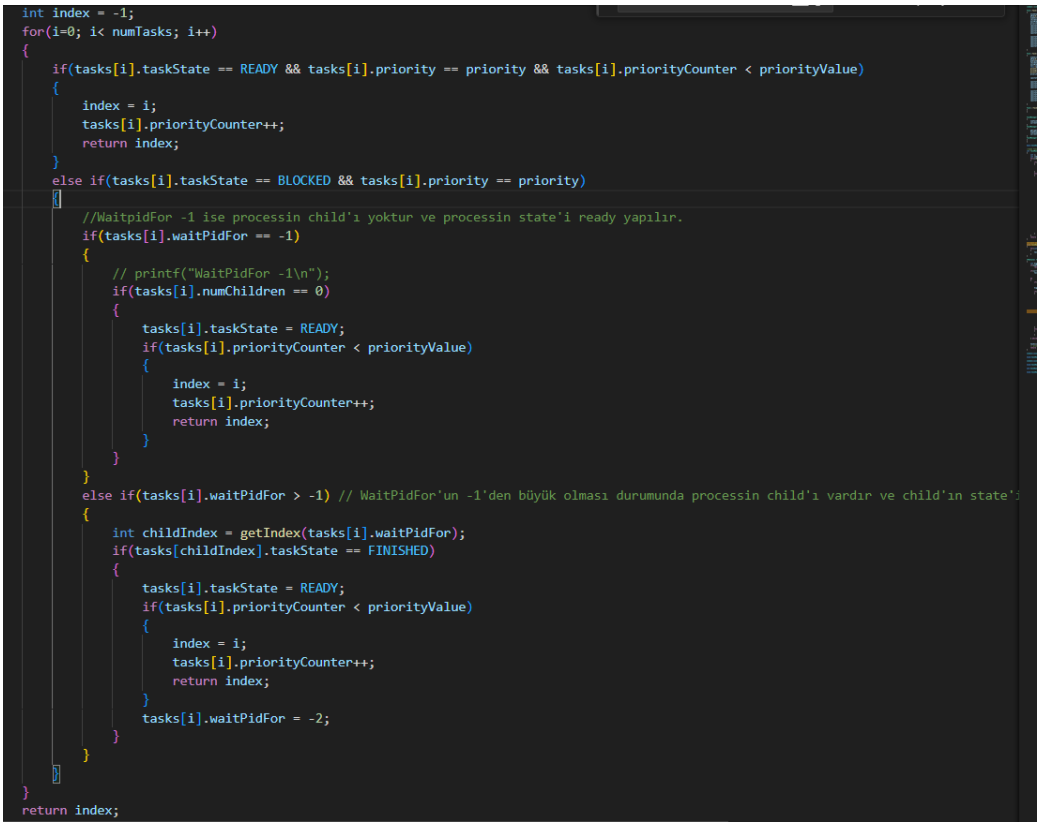
There are mainly three priority which are High, Medium and Low. They are represented as enum.

```
enum TaskState {READY,RUNNING, BLOCKED, FINISHED};
typedef enum { Low, Medium, High } TaskPriority;
```

To the Task class I added priority and priortiy counter varaibles. Priority represents the priority of the process and priority counter represents the number of times the process will run consecutively. And then I changed the constructors. I set these two new variables. For example for five time interrupts High process will be executed and then for three interrupts Medium process will be executed and then for one interrupt low level process will be executed.

To do this I changed my scheduler. First of all I implement a function wihich search all the task array linearly. According to given priority argument, it checks all the array. If the priority of the argument matches with the process's priority, it checks the priority counter of the process. For the High priority counter supposed to be less than 5, fort he medium priority is supposed to be less than 3 and fort he low it supposed to be less than 1.

For the lifecycle 1 which uses round robin, I used my new scheduler algorithm. If I give the tasks same priority, It will work as regüler round robing algorithm. So I will give the init process High priority and the all the forked children have High priority. Then it will be executed by using regüler round robin.



```
int index = -1;
for(i=0; i< numTasks; i++)
{
    if(tasks[i].taskState == READY && tasks[i].priority == priority && tasks[i].priorityCounter < priorityValue)
    {
        index = i;
        tasks[i].priorityCounter++;
        return index;
    }
    else if(tasks[i].taskState == BLOCKED && tasks[i].priority == priority)
    {
        //WaitpidFor -1 ise processin child'ı yoktur ve processin state'i ready yapılır.
        if(tasks[i].waitPidFor == -1)
        {
            // printf("WaitPidFor -1\n");
            if(tasks[i].numChildren == 0)
            {
                tasks[i].taskState = READY;
                if(tasks[i].priorityCounter < priorityValue)
                {
                    index = i;
                    tasks[i].priorityCounter++;
                    return index;
                }
            }
        }
        else if(tasks[i].waitPidFor > -1) // WaitPidFor'un -1'den büyük olması durumunda processin child'ı vardır ve child'ın state'
        {
            int childIndex = getIndex(tasks[i].waitPidFor);
            if(tasks[childIndex].taskState == FINISHED)
            {
                tasks[i].taskState = READY;
                if(tasks[i].priorityCounter < priorityValue)
                {
                    index = i;
                    tasks[i].priorityCounter++;
                    return index;
                }
                tasks[i].waitPidFor = -2;
            }
        }
    }
}
return index;
```

In the scheduler, first of all it calls this function with High and 5 parameters. If it can find a process which has a proper priority and priority counter, it return the index of the process and it assignes it to the currentTask. If it can not find proper process, it returns -1 which means there are not any High level processes or All the High level processes are executed 5 times. This indicates that it is the turn of the medium priority. it calls this function with Medium and 3 parameters. If it can find a process which has a proper priority and priority counter, it return the index of the process and it assignes it to the currentTask. If it can not find proper process, it returns -1 which means there are not any Medium level processes or all the medium level processes are executed 3 times. This indicates that it is the turn of the low priority. It calls this function with Low and 1 parameters. If it can find a process which has a proper priority and priority counter, it return the index of the process and it assignes it to the currentTask. If it can not find proper process, it returns -1 which means there are not any Low level processes or all the Low level processes are executed 1 times. And since there are not any available processes, it calls setPriorityCountersZero function and it sets all the priority counters to zero.

It means all the processes are available to be executed according to their priorities.

```
//I will use this function to reset priority counters when all tasks are blocked.
void TaskManager::setPriorityCountersZero()
{
    for(int i=0; i< numTasks; i++)
    {
        tasks[i].priorityCounter = 0;
    }
}
```

```
do
{
    if(tasks[currentTask].taskState == RUNNING)
        tasks[currentTask].taskState = READY;

    highValue = linearSearchForPriority(High, 10);

    if(highValue == -1)
    {
        mediumValue =  linearSearchForPriority(Medium, 5);

        if(mediumValue == -1)
        {
            lowValue = linearSearchForPriority(Low, 1);

            if(lowValue == -1)
            {
                // printf("All tasks are blocked. Resetting priority counterszozozozoo.\n");
                setPriorityCountersZero();
                lowValue = 0;
            }
            else
                currentTask = lowValue;
        }
        else
            currentTask = mediumValue;
    }
    else
    {
        currentTask = highValue;
    }

} while (lowValue == -1);
```

This is for lifecycle1:

```
Task taskKernel1(&gdt, initKernel1, High);
taskManager.AddTask(&taskKernel1);
```

It uses randNum to generate random variable. Then it forks ten times and according to random variable it calls a function.

```
void initKernel1()
{
    my_srand(12345);
    int randomNumber = randNum(1,4)%4;
    while(randomNumber == 0)
    {
        randomNumber = randNum(1,4)%4;
    }

    printf("Random number: ");
    printDigit(randomNumber);
    printf("\n");
    // int randomNumber =3 ;

    uint32_t pid=0;
    uint32_t child=-1;
    for(int i=0;i<10;i++)
    {
        child = myos::fork();
        if(child==0)
        {
            if(randomNumber==1)
            {
                TaskBinarySearch();
            }
            else if (randomNumber==2)
            {
                TaskLinearSearch();
            }
            else if(randomNumber == 3 )
            {
                TaskForkCollatz();
            }
            else if( randomNumber == 4)
            {
                TaskForkLongRunningProgram();
            }
            exit_process();
        }
    }
    wait_process(-1);
    printf("All child processes are finished.\n");
    while(true);
}
```

Parent process wait for all children.

Random number is two: so linear search:

```
Random number: 2
Output: -1
Output: -1
Output: -1
Output: -1
Output: -1
Output: -1
Output: -1
Output: -1
Output: -1
Output: -1
All child processes are finished.
```

It can not find and returns -1 for each process.

```
void TaskLinearSearch(){
    int arr[] = {15, 25, 35, 55, 60, 80, 105, 115, 135, 175};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 170;
    int result = linear_search(arr, n, key);
    printf("Output: ");
    printDigit(result);
    printf("\n");
    // exit_process();
}
```

This is output with process table:

```
pId   pPid      State   Priority    CurrentTask
1      0       Blocked     High          4      0
2      1       Finished     High         4      7
3      1       Finished     High         4      9
4      1       Finished     High         4      4
5      1       Running     High          4      9
6      1       Ready       High          4      0
7      1       Ready       High          4      0
8      1       Ready       High          4      0
9      1       Ready       High          4      0
10      1       Ready       High         4      0
11      1       Ready       High         4      0
Output: -1
```

Random number with 1: Binary Search:

```
Random number: 1

Output: -1

Output: -1

Output: -1

Output: -1

Output: -1

Output: -1

Output: -1

Output: -1

Output: -1

Output: -1

Output: -1
All child processes are finished.
```

```c
void TaskBinarySearch(){
    int arr[] = {15, 25, 35, 55, 60, 80, 105, 115, 135, 175};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 110;
    int result = binary_search(arr, 0, n - 1, key);
    printf("\nOutput: ");
    printDigit(result);
    printf("\n");

    //exit_process();
}
```

Random number generator:

```
uint32_t my_rand() {
    // LCG parametreleri
    asm("rdtsc": "=A"(seed));
    return seed;
}

// Rastgele sayı üreteciyi tohumlayan fonksiyon
void my_srand(uint32_t s) {
    seed = s;
}

// Verilen aralıkta rastgele bir sayı döndüren fonksiyon
int randNum(int min, int max) {
    return my_rand() % (max - min + 1) + min;
}
```

For lifecycle2:

```
Task taskKernel2(&gdt, initKernel2, High);
taskManager.AddTask(&taskKernel2);
```

```
Random number1: 2 Random number 2: 3
Output: -1
Output: -1
Output: -1
3 : 10 5 16 8 4 2 1
2 : 1
3 : 10 5 16 8 4 2 1
2 : 1
3 : 10 5 16 8 4 2 1
2 : 1
All child processes are finished.
```

According to random numbers, processes are linear search and collatz.

Fort he priority adjustment:

```
 Task task1(&gdt, taskA, High);
 Task task2(&gdt, taskB, Medium);
 Task task3(&gdt, taskC, Low);
 // Task task4(&gdt, taskD);
 // Task task5(&gdt,execTestExamle);
/*  Task task6(&gdt,TaskForkCollatz);
 Task task7(&gdt,TaskForkCollatz);
 Task task8(&gdt,TaskForkCollatz);
 Task task9(&gdt,TaskForkLongRunningProgram);
 Task task10(&gdt,TaskForkLongRunningProgram);
 Task task11(&gdt,TaskForkLongRunningProgram);


 taskManager.AddTask(&task1);
 taskManager.AddTask(&task2);
 taskManager.AddTask(&task3);
```

```
uint32_t time = 10000000;

void taskA()
{
    int i=0;
    while(true)
    {
        if(i%time == 0)
        {
            sysprintf("A ");
            // exit_process();
        }
        i++;
    }
}

void taskB()
{
    int i=0;
    while(true)
    {
        if(i%time == 0)
            printf("B ");
        i++;
    }
}
```

Since TaskA has high priority i prints 5 times more than C. And TaskB prints 5 times more than C.

```
A A A A A A A A A A A A A A A A A A A A A A A A A A A B B B B B B B B B B B B B
B C C C C C C A A A A A A A A A A A A A A A A A A A A A A A A A A B B B B B B B
B B B B B B C C C C C A A A A A A A A A A A A A A A A A A A A A A A A A A A B B
B B B B B B B B B B B C C C C C C A A A A A A A A A A A A A A A
```