OperatingSystemDesign

Implementation assignment #2

19102091-YoungHwanPhan

```java
public class PIDTester {

    public static void main(String[] args) {
        PIDManagerClass pidM = PIDManagerClass.getInstance();
        Scanner scan = new Scanner(System.in);
        System.out.println("--------------ITM-19102091-YoungHwan-Phan---------------");
        System.out.println("---------------------------------------------------------");
        System.out.println("--------------INPUT-POSTIVE-INTEGER-NUMBER--------------");
        System.out.println("---------------------------------------------------------");
        System.out.println("-----------ThreadNum, ThreadTime, ProcessTime-----------");
        try {
            int ThreadNum = scan.nextInt();
            int ThreadTime = scan.nextInt();
            int ProcessTime = scan.nextInt();
            if(ThreadNum<0||ThreadTime<0||ProcessTime <0) {
                System.out.println("<<<<Please input 'POSITIVE' 'INTEGER' number>>>>");
                System.out.println("<<<<Restart Program please>>>>");
                System.exit(0);
            }
            System.out.println("-----Select Mode : 0.getPID(), OtherNum.getPIDWait()------");
            int pidMode = scan.nextInt();
            pidM.setPIDManager(ThreadNum, ThreadTime, ProcessTime, pidMode);
        }catch(Exception e) {
            System.out.println("<<<<Please input 'POSITIVE' 'INTEGER' number>>>>");
            System.out.println("<<<<Restart Program please>>>>");
        }finally {
            scan.close();
        }


    }


}
```

If user input unexpected type number (float or negative int), program notice this num is not available. And recommend restart program.

```java
public class PIDTester {

    public static void main(String[] args) {
        PIDManagerClass pidM = PIDManagerClass.getInstance();
        Scanner scan = new Scanner(System.in);
        System.out.println("--------------ITM-19102091-YoungHwan-Phan---------------");
        System.out.println("---------------------------------------------------------");
        System.out.println("--------------INPUT-POSTIVE-INTEGER-NUMBER--------------");
        System.out.println("---------------------------------------------------------");
        System.out.println("-----------ThreadNum, ThreadTime, ProcessTime-----------");
        try {
            int ThreadNum = scan.nextInt();
            int ThreadTime = scan.nextInt();
            int ProcessTime = scan.nextInt();
            if(ThreadNum<0||ThreadTime<0||ProcessTime <0) {
                System.out.println("<<<<Please input 'POSITIVE' 'INTEGER' number>>>>");
                System.out.println("<<<<Restart Program please>>>>");
                System.exit(0);
            }
            System.out.println("-----Select Mode : 0.getPID(), OtherNum.getPIDWait()------");
            int pidMode = scan.nextInt();
            pidM.setPIDManager(ThreadNum, ThreadTime, ProcessTime, pidMode);
        }catch(Exception e) {
            System.out.println("<<<<Please input 'POSITIVE' 'INTEGER' number>>>>");
            System.out.println("<<<<Restart Program please>>>>");
        }finally {
            scan.close();
        }


    }



}
```

User can test getPID() version PIDManager as input 0.
If want to test getPIDWait(), just input positive number except 0.

```java
public class PIDTester {

    public static void main(String[] args) {
        PIDManagerClass pidM = PIDManagerClass.getInstance();
        Scanner scan = new Scanner(System.in);
        System.out.println("--------------ITM-19102091-YoungHwan-Phan---------------");
```

```java
public class PIDManagerClass implements PIDManager {
    private static boolean flag = true;
    private static PIDManagerClass instance = new PIDManagerClass();
    private Vector<Integer> pids = new Vector<>();

    private PIDManagerClass() {

    }

    public static PIDManagerClass getInstance() {
        return instance;
    }
```

And PIDManager must exist only one. For prevent duplicate creating manager, I make pidmanagerclass constructor as private, and make this instance at its global variable space as private.

Other class only connect this instance by getInstance() method.

```
public class PIDManagerClass implements PIDManager {
    private static boolean flag = true;
    private static PIDManagerClass instance = new PIDManagerClass();
    private Vector<Integer> pids = new Vector<>();

    private PIDManagerClass() {

    }

    public static PIDManagerClass getInstance() {
        return instance;
    }
```

I declare pid container as Vector. Many thread objects want to connect pid container by getPID or getPIDWait.

By using vector, I can prevent duplicate pid when thread objects arrive container at same time.

And also, other class must not connect this container directly. So I set this as private.

The reason why declared as private next time is all the same reason, so I will not mention it after.

PIDManagerClass

```
public int setPIDManager(int threadNum, int threadTime, int processTime, int getPIDType) {
    System.out.println("----------------PID MANAGER SET----------------");

    for(int i = MIN_PID; i <= MAX_PID; i++) {
        pids.add(i);

    }
    for(int i = 0; i <= threadNum; i++) {


        new MyThread(("thread"+(i)),threadTime,processTime, getPIDType);
    }

    return 1;
}
```

Before user want to test PIDManager, manager need some data about test.

User input thread number, thread running time, processTime, getPIDtype.

And then, initiate pid container and make thread object.

PIDManagerClass

```java
@Override
public int getPID() {
    if(pids.isEmpty()) {
        return -1;
    }else {
        int pd = pids.get(0);
        pids.remove(0);
        return pd;
    }

}
```

getPID() : if pid container is empty, return -1. if not, return available pid

```java
@Override
public int getPIDWait() {
    while(!flag) {
        //waiting
    }
    flag = false;
    //Critical Section
    int pidchild = getPID();
    while(pidchild==-1) {
        System.out.println("wait...");
        try {
            Thread.sleep(500);
        }catch(Exception e) {

        }
        pidchild= getPID();

    }
    flag = true;
    //Critical Section
    return pidchild;
}
```

getPIDWait() : I watch some error. That is unexpected pid(ex 0,1,,, such that is smaller than MIN_PID declared at interface) is released to pid container. So I declare flag as initiate true.

If getPID() return -1(no available pid) wait in the while. And update pid every time in the while. If available pid exist, get out the while and set flag as true. And return available pid.

PIDManagerClass

```
@Override
public void releasePID(int pid) {
    try {
        pids.add(pid);

    }catch(IllegalArgumentException e) {
        System.err.println(e);
    }

}
```

If process return pid to pid container,
Just add that into container

```java
public class MyThread extends Thread {
    private String threadName;
    private int createTime;
    private int threadTime;
    private int processTime;

    private int pid;
    private int getPIDType;
    private Thread runningThreadasProcess;
    private Random random = new Random();
    private PIDManagerClass pidM = PIDManagerClass.getInstance();

    public MyThread(String threadName, int threadTime,int processTime, int getPIDType) {
        this.threadName = threadName;
        this.threadTime = threadTime;
        this.processTime = processTime;

        this.getPIDType = getPIDType;
        this.runningThreadasProcess = new Thread(this, this.threadName);

        runningThreadasProcess.start();
    }
```

Thread global variable field

And I understand thread is run as process and get pid. So I declare running thread as runningThreadasProcess.

```java
@Override
public void run() {
    getPIDtype(this.getPIDType);

}
```

I make getPIDtype method because code that is in run() is too long

Test getPID() version PIDManager .

I implemented the time when thread was generated as a wake-up call as soon as thread was created.

```
public void getPIDtype(int gettype) {
    if(gettype == 0) {
        try {
            createTime = random.nextInt(processTime*1000);
            Thread.sleep(createTime);
            this.pid= pidM.getPID();
            if(this.pid !=-1) {
                System.out.println(threadName+" created at "+createTime+"ms "+"pid: "+this.pid);

            }else {
                //if getPID() return -1, this thread is covered under else part by return;
            }
        }catch(Exception e) {
            System.err.println(e);
        }

        try {
            if((createTime+threadTime*1000) > processTime*1000) {
                Thread.sleep(threadTime*1000);
                System.out.println(processTime+"sec passed... Program ends");
                System.exit(0);
            }else {
                if(this.pid == -1) {
                    System.out.println("All pid are used now.");
                    System.out.println("this thread can not get pid.");
                    return;
                }
                Thread.sleep(threadTime*1000);
                pidM.releasePID(this.pid);
                System.out.println(threadName+" destroyed at "+(createTime+threadTime*1000)+"ms"+" pid: "+this.pid );
            }
        }catch(Exception e) {
            System.err.println(e);
        }
```

If thread can not get pid, just drop it as return;

If not , thread sleep as running time and release its pid.

getPIDWait()

```
    }else {
        try {

            createTime = random.nextInt(processTime*1000);
            Thread.sleep(createTime);
            this.pid= pidM.getPIDWait();
            if(this.pid !=-1) {

                System.out.println(threadName+" created at "+createTime+"ms "+"pid: "+this.pid);


            }else {
                //getPIDWait() cover this part.
            }
        }catch(Exception e) {
            System.err.println(e);
        }

        try {

            if((createTime+threadTime*1000) > processTime*1000) {
                Thread.sleep(threadTime*1000);
                System.out.println(processTime+" sec passed... Program ends");
                System.exit(0);
            }else {
                Thread.sleep(threadTime*1000);
                pidM.releasePID(this.pid);
                System.out.println(threadName+" destroyed at "+(createTime+threadTime*1000)+"ms"+"pid: "+this.pid );
            }
        }catch(Exception e) {
            System.err.println(e);
        }
    }

}
```

Unlike getPID(), getPIDWait method handles the case that there is no available pid, so there were not codes for that part.

Thank you!