**Programming Assignment #2**

1. **Introduction**

* **What you will talk about/do**

This Project is designing a pid Manager that is responsible for managing pids. When a process is first created, it is assigned a unique pid by the pid manager. The pid is returned to the pid manager when the process completes execution.

* **Overview of the rest of your report**

Using Java Interface, the basic method for obtaining and releasing a pid is identified. The difference between getPID() and getPIDWati() is that if no pids are available, getPID() returns -1, whereas getPIDWati() blocks the calling process until a pid becomes available. These two methods can be selected by the user to output the results. It will explain the program executed by inputting the number of threads, the lifetime of the program, and the lifetime of the thread to the user.

* **Project**
  + **Your approach to the problem**

Let’s assume 4 cases to see if this program runs well.

**Case1**: Number of threads less than pid range, using getPID()

**Case2**: Number of threads more than pid range, using getPID()

**Case3**: Number of threads less than pid range, using getWaitPID()

**Case4**: Number of threads less than pid range, using getWaitPID()

I'm trying to make a program that works well in all cases.

* + **What you did**

The pidmanager interface was implemented, and functions were implemented by overriding each. Also, I used the runnable interface for thread generation. I will explain the coed in detail later.

* + **Design**
    - What you already had (and where it came from)

I implemented the PIDManager interface in the book.

Text

Description automatically generated

* + - What you added/changed
      * + PIDManager interface implements

Global variables

startTime: represent the start time

lifetimeProgram: lifetime of program to be inputted by user

lifetimeThread: lifetime of thread to be inputted by user

cur: the current pid

pidArr: to represent flag(0: not using, 1: using) of pid

semaphore: determine wait() in getPIDWait().

select: (1) getPID() / (2) getPIDWait() selected by user

Text

Description automatically generated

num: thread number

the constructor Test()

obj: to use the monitor(synchronized)

Graphical user interface, text

Description automatically generated with medium confidence

getPID()

if the pid is in the range, flag is changed 1 and pid +1

however, the return value is origin pid value

Text

Description automatically generated

getPIDWait()

to use monitor, using synchronized.

If no pids are available, the program is wait until pid become available.

Likewise getPID(), if the pid is in the range, flag is changed 1.

Text

Description automatically generated

releasePID()

After using the pid, change the flag to zero.

Then, wakes up waiting thread using notify()

Text

Description automatically generated

Thread

Override the run()

if you select getPID(), you can use getPID(). In this case, if no pids are available, you can see the results "No pids are availbale".

if you select getPIDWait(), you can use getPIDWait(). In this case, if no pids are available, you will wait unit pids are available.

Use the variable differ to show the execution time.

Use sleep() to sleep during execution time. Multiply by 1000 for unit conversion.

Text

Description automatically generated

Main()

Input the number of threads, the lifetime of the program, and the lifetime of the thread

Select the method getPID() or getPIDWait()

Text

Description automatically generated

Using currentTimeMills() represent time

To create the thread using for loop

Making the thread using runnable class, then resist the thread.

To destroy the thread using while loop

If the program is finished, print the finish sentence

Text

Description automatically generated

* + Results
    - Input

Text

Description automatically generated

* + - **Case1**: Number of threads less than pid range, using getPID()
      * No wait

Text

Description automatically generated

* + - **Case2**: Number of threads more than pid range, using getPID()
      * No wait and print no pids are available

Text

Description automatically generated

A picture containing text

Description automatically generated

…

Text

Description automatically generated

* + - **Case3**: Number of threads less than pid range, using getWaitPID()
      * No wait

Text

Description automatically generated

* + - **Case4**: Number of threads less than pid range, using getWaitPID()
      * No wait: when the thread lifetime is short

Text

Description automatically generated

A picture containing text

Description automatically generated

* + - * Wait

Text

Description automatically generated

Text

Description automatically generated

* Can see the wait(62 – 65)

1. **Conclusions**

In fact, it became easier to understand by directly implementing the synchronization learned as a concept. In addition, in order to implement each function, the concept could be reconsidered, understood, and implemented by referring to the example codes in the book.

The most memorable parts are the implementation of getPID() and getPIDWait(). I understood better by implementing the semaphore and monitor that I had studied myself.

When I thought about why these results came out while doing the assignment, I could understand if I thought about the concept again.