LAB 5 - OS

UDAY KUSUPATI - 140050048
NIKHIL KUMAR - 140050037

1.

The priority can be any positive integer ranging from 1 to infinity, when the user tries to set the priority to some value which is less than or equal to zero, we don't change the priority and leave the priority as the previous priority value.(The return value of setprio is set to -1 to notify the user the same). On successful setting of the priority value, we return 0. Note that we set the default value of the priority to 50 in allocproc

2.

Interpretation of the priority value and the way we schedule:

In the long run, any two processes X, Y with priorities x, y get scheduled in the ratio x:y, i.e we are implementing a weighted round robin policy.

Kernel Data Structures which are added:

We have added an int variable named cycles_run in the struct proc , which is initialised to 0 at the time of process creation.(This is of course in addition to the int variable priority which is used for part 1)

IDEA: I will run a process only if it has not been run for the priority number of cycles, and when all runnable processes have been run for priority number of cycles, (we get that from the above criteria we cannot be able to run any process). So I will set the number of cycles run of all the runnable processes to zero.

IMPLEMENTATION:

We have modified the existing scheduler in the following manner.

We first set a variable(which is a member of struct proc) cycles_run to zero at the time of process creation. Also increment just before you context switch into that process, i.e just before calling swtch function in the scheduler function.

And we only let a process which is runnable to run if it's cycles_run < priority.

Maintain a variable ran which says that in a particular iteration over all the processes, whether I was able to run any process or not. If I am not able to run any process, then set all the runnable process's cycles_run value back to Zero.

The run time of this is approximately O(1)**.**

(Reason: When all the processes be runnable state, then we will have to do that setting of the variable cycles_run to 0 once in every x cycles, where x is the sum of priorities of all cycles which is generally large because default priority is set to a large number(50), so it is not exactly O(1), but it is more than O(1) by NPROC/x which is very small because NPROC here is 64 and x is very large. So we can safely consider that our scheduler runs in O(1) time.)

3. We set the default value of priority to 50 before someone set's it.
If a process which is scheduled, it's state will be set to WAITING, which will not be scheduled until it's state becomes RUNNING, so we will thus not be keeping the CPU idle.
Yes, our code is indeed safe to run over multiple CPU cores, because we have aquired ptable.lock and released it appropriately whenever we are updating a shared variable.(variables from struct proc)

4.
CPU BOUND PROCESS DEMONSTRATION:
Consider the code testmyscheduler,
The normal scheduler gives equal number of "child" and "print" approximately at any given instant, and both terminate at approximately
Our scheduler does it differently, the priorities of parent and child are set to 20, 40 respectively. So at any instant the number of parent's printed and child's printed will be approximately in the ratio 1:2, the child terminates first after printing "child" 20 times. Till now the parent has been printed 10 times(20/2). Then the parent gets printed for the remaining 10 times.

I/O BOUND PROCESS DEMONSTRATION AND BLOCKING CALL DEMONSTRATION:

1. I have put the output of this at the end of the report testmy2 .I am trying to create a file once and read from it many times.It keeps scheduling it based on the read call and availability of the file in the cache.
2. I am doing blocking calls in parent and child, in testmy3. The output is put at the end of the report. As i am doing blocking calls, it keeps calling scheduler, and gets context switched out as it is not runnable even though one has more priority over the other. So we get parent and child alternatively.

CHANGES MADE:

If you want to find where all changes are made, look for this line “EDIT MADE HERE” in the following files.

- syscall.c
- syscall.h
- sysproc.c
- user.h
- usys.S
- defs.h
- proc.c
- proc,h

And also I have changed two lines in the Makefile, added the user program name in UPROGS, EXTRA

REAMDE:

I have added 3 user programes to demonstrate the correctness of myscheduler. testmyscheduler, testmy2, testmy3 . These are put in the folder and the makefile is appropriately changed

OUTPUTS FOR TEST CASES

OUTPUT for testmyscheduler:

Child
Parent
Child
Parent
Child
Child
Child
Child
Parent
Child
Parent
Child
Parent
Child
Child
Child
Child
Parent
Child
Parent
Child
Parent
Child
Child
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Parent
Parent
Parent
Parent

Parent
Parent
Parent
Parent

OUTPUT for testmy2:

small file test
creat small succeeded; ok
writes ok
The priority of the parent is 50
small file test
creat small succeeded; ok
writes ok
The priority of the child is 100
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded okread succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
 in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child

read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
read succeeded ok in child
The child has terminated
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent

read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
read succeeded ok in parent
The parent has terminated

OUTPUT FOR testmy3:

Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent

Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child
Parent
Child