Chris Kasper
CS 370
Programming Assignment 2

Design

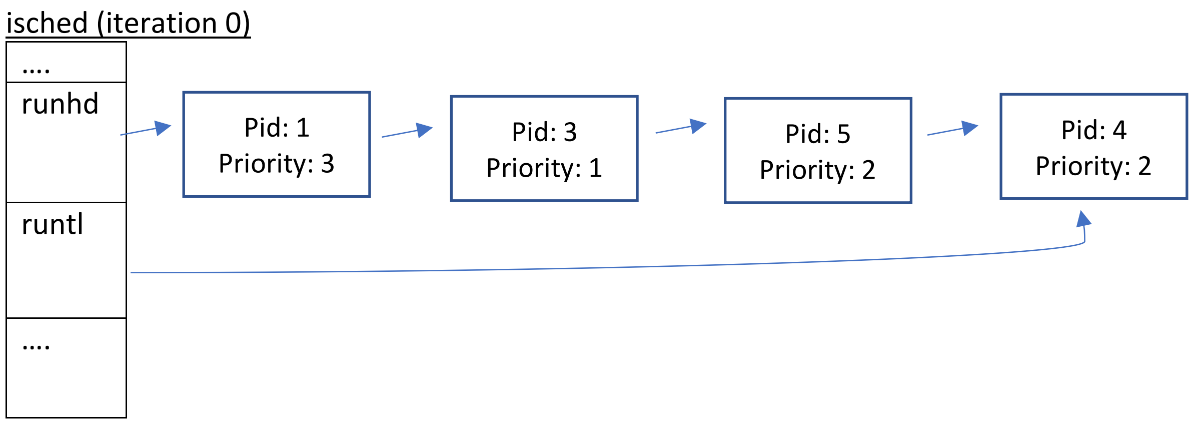
The scheduler I decided to implement is a UNIX-style priority scheduler with priority adjustments to favor interactive processes, with increases in quanta.

This was implemented by first adding a priority variable to the *Prog* data structure. There are three different levels or priority: 1, 2, and 3. The higher the number, the higher the priority. In addition to the higher priority level, these processes get more quanta. The formula for calculating quanta for a priority level is $p' = PQUANTA - (1000 * (p - 1))$. As you can see, the changes in quanta are by 1000.

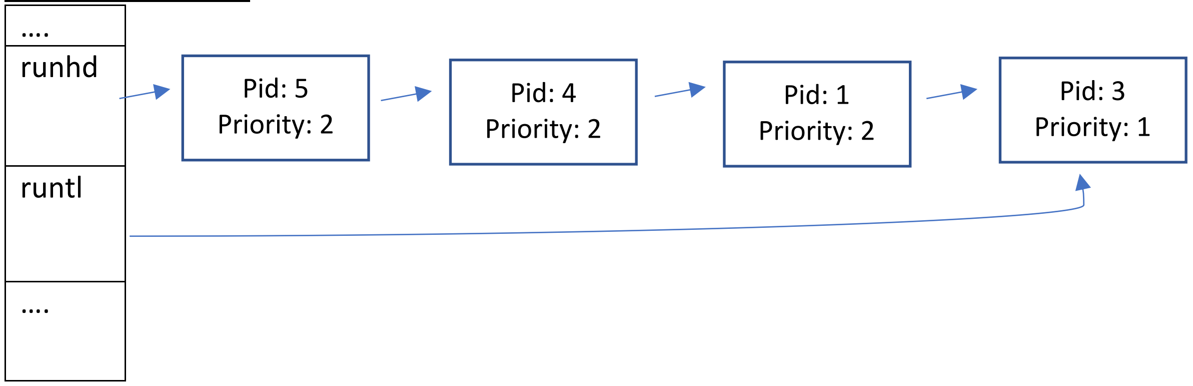
Right in the beginning, processes are set to a baseline priority of 2. This priority value gets adjusted depending how its next time slice goes. If the process enters one of the three blocked states (Precv, Psend, Palt), the priority of the process gets incremented. However, if the process does not get blocked during its time slice, therefore leaving the CPU in the ready state, its decreases its priority. The benefit of this strategy is that new processes get a medium priority right off the bat, and that interactive processes get the highest priority to the scheduler.

The original Inferno scheduler code utilizes a round-robin system, where each process takes turns on the CPU with the same amount of quanta. Once the process at the head of the list finishes, it goes to the back of the list. While we are changing the scheduler, I was able to pretty much use the same code that it uses and not have to change the data structure of *isched*. Right after a process gets sent to the back to the list, I use a function *getHighestPriority*(), to find the highest current priority on the ready list of processes. When the priority value is returned, I iterate through the list of the ready processes until I find a process that matches the priority level.

Here is an illustration of the new scheduler:

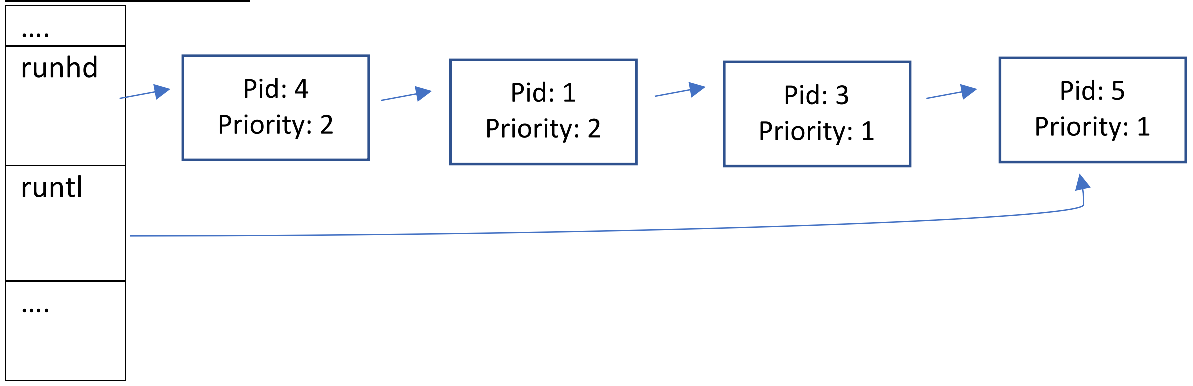


isched (iteration 1)



Note: Process 1 went through CPU without getting blocked (therefore, decrease in priority).

isched (iteration 2)



Note: Process 5 went through CPU without blocked (therefore, decrease in priority).

Going off the figures, you can see that the highest priority processes are scheduled in a round robin style.

Testing

For testing purposes, I created a compute bound process (in Limbo) that computes the Fibonacci Sequence. This program took an integer input and would return the last number in that sequence (i.e. ./myFib.dis 4 returns 3).

Here is the code for the program:

```
implement myFib;

include "sys.m";
include "draw.m";

myFib: module
{
	init: fn(ctxt: ref Draw->Context, args: list of string);
};

init(ctxt: ref Draw->Context, args: list of string)
{
	sys := load Sys Sys->PATH;

	numStr : string;

	for (; args != nil; args = tl args) {
		numStr = hd args;
	}

	numInt := int numStr;
	ans := fib(numInt);

	sys->print("myFib(%d) = %d\n", numInt, ans);
}

fib(n: int): int {
	if (n == 0) {
		return 0;
	}
	if (n == 1) {
		return 1;
	}
	return fib(n-1) + fib(n-2);
}
```

As for the actual testing, what I did was run two Fibonacci Sequence processes at the same time, while running two Tetris games (actively playing one and restarting the other when needed) from the Inferno GUI. Timing just two Fibonacci sequences between the two schedulers doesn't show much difference, since there are not any interactive processes occurring too, which is why its omitted from the report.

Here are the time trials from the testing:

Old scheduler

Time #	**Time (in seconds)**
1	147.9
2	149.2
3	149.3
4	148.5
5	150.1
Average Time	149

New priority scheduler

Time #	**Time (in seconds)**
1	154.9
2	152.9
3	153.2
4	154.2
5	152.3
Average Time	153.5

From the time trials, we can infer that my priority scheduler devotes more time to the interactive processes because the two computations take on average about 4.5 seconds longer than they do with the original round robin scheduler.