Name: I-Hsien Huang

[BACKGROUND]
AOS XV6

- **sysproc.c** add the real implementation of your method here
- **syscall.h** define the position of the system call vector that connect to your implementation
- **user.h** define the function that can be called through the shell
- **syscall.c** external define the function that connect the shell and the kernel, use the position defined in
- **syscall.h** to add the function to the system call vector
- **usys.S** use the macro to define connect the call of user to the system call function
- **defs.h** add a forward declaration for your new system call
- **sysfunc.h** add the new system call handler into this file too like "int sys_newsystemcall(void)
- **Proc.c** record the function what you want to do

info(parameter): 1,2,3
(1) A count of the processes in the system;
-> look into the ptable (recorded the processes) and find out (~UNUSED)
(2) A count of the total number of system calls that a process has done so far
-> in proc.h declare a parameter to keep the number, and every time when the system call Syscall, take a record.
(3) The number of memory pages the current process is using.
-> in proc.h there is a parameter to record process size (sz) , and divide it by the page size.
1)

```
1 sleep  init
2 sleep  sh
3 run    test


try to get count of processes 3
```

2) rand is a system call

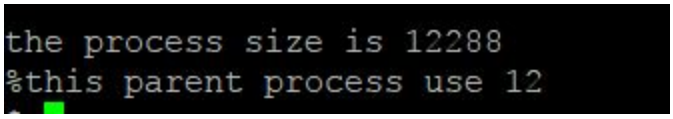
```
use 13 system calls
rand() = 195773072hello from the kernel space!

rand() = -1944058078hello from the kernel space!

rand() = 1973987086hello from the kernel space!

use 17 system calls
```

3) pagenum = process_size(sz) *4 / PGSIZE



[note]
How to pass a parameter into a syscall?
-> use argint(0, &parameter) to read the parameter passed by the user program.
(called in the syscall.c)
How to add a system call?

1. usys.S: add the system call
2. Syscall.h: define the syscall num
3. syscall.c :
4. Sysproc.c: call the function u write in the proc.c
5. Proc.c: write your code into the function
6. Def.h: add a forward declaration for your new system call
7. User.h: define the function that can be called through the shell

//----------

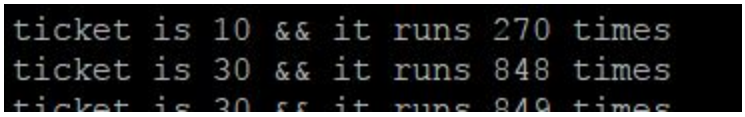
[note]
How to decide the CPU number to 1? (recorded in the Makefile)
-> make CPUS=1 qemu-nox

Normally, scheduling in xv6 is to search into the ptable and find out the state is RUNNABLE
If there are three processes A.B.C, run at the same time, it will find the first process and run it, and after a time ticker, keep searching in the ptable and find another RUNNABLE process.
Therefore, the sequence will be A.B,C,A,B,C,A,B,C,A,B,C,A,B,C……..

//to decide what kind of scheduling the user want
#define STRIDE 0
#define LOTTERY 1

LOTTERY SCHEDULING:
In the Lottery Scheduling. Every process has a given tickets, and every time scheduling the system will generate a random number between 0 ~ toal_tickets (sum the tickets of RUNNABLE processes).
For example:
A: tickets = 10, B: 20, C:30; and assume the sequence in the ptable is A,B,C.
Hence, when the system generate the number 0~10: A will be chosen, 11~30, B: will be chosen...

[TEST] prog 1&;prog 3
1 means tickets is 10;
3 means tickets is 30;
Consequently, we can make sure that the scheduling is correct, for the reason that the ratio of running times is consistent with the ratio of the tickets number.



]

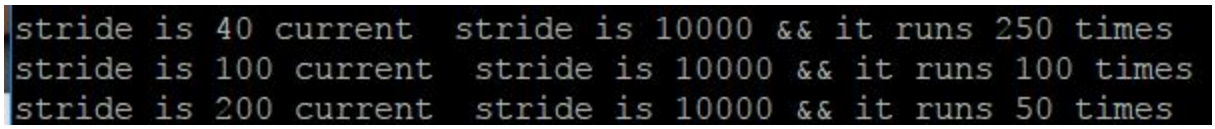
[TEST] prog 1&;prog 2&;prog 3 229 : 470 : 711 => 1: 2.05 : 3.1 (accurate enough)

STRIDE SCHEDULING:

The concept is every process has its own tickets, and the stride is COSTANT/tickets. That means that the more tickets the process have, the less stride it has. And every time scheduling, search into all RUNNABLE processes, and run the min cur_stride. The chosen process will add stride to the cur_stride.

Stride is const but the accumulation stride is not.

[TEST]
prog 10&; prog 5&; prog 25&
prog 100 tickets -> stride 100
prog 50 tickets -> stride 200
Prog 250 tickets -> stride 40



proc.c

Additional note: in xv6, the “%”, the compiler can not figure out L value and R value about the “%”. Therefore we need “()”.

[more complete evaluation of the schedulers]
Actually, when we only compare the time efficient for the normal scheduling and stride or lottery. The efficiency of Stride and Lottery is low, since we add some calculations and they takes time. However, the main purpose of Stride and Lottery is similar to priority scheduling, run the most hurry process, but it still run other processes (that’s the difference between priority process and Stride and Lottery)
Moreover, when it comes to compare Stride and Lottery.
In the long-term, both of these are similar.

In the short-term, Stride is more stable since Lottery include some probability, and Stride is directly decided by the min of cur_stride. There are some rules and sequence in the Stride.

Cite:
https://stackoverflow.com/questions/8021774/how-do-i-add-a-system-call-utility-in-xv6?rq=1