CSL301
Assignment 2
Amay Dixit - 12340220

Step 1: Update `proc.h` – Add Priority Field

File: `proc.h`
Code Added:

```
int priority;
```

Explanation:
Added an integer field `priority` to the process control block to store each process's static priority. Lower numbers indicate higher priority.

Step 2: Initialize Priority in `proc.c`

File: `proc.c` – inside `allocproc()`
Code Added:

```
p->state = EMBRYO;
p->pid = nextpid++;
p->priority = 50; // added this line
```

Explanation:
Assigned a default priority of 50 to all new processes when allocated.

Step 3: Implement Priority Scheduler in `proc.c`

File: `proc.c` – replace `scheduler()` function
Code Added:

```
void
scheduler(void)
{
 struct proc *p;
 struct cpu *c = mycpu();
 c->proc = 0;
  for(;;){
```

```
  // Enable interrupts on this processor.
  sti();
  struct proc *highest_priority_p = 0;
  int highest_priority = 1000; // high initial value
  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state != RUNNABLE)
      continue;

    if (p->priority < highest_priority) {
        highest_priority = p->priority;
        highest_priority_p = p;
    }
  }

  if (highest_priority_p != 0) {
      p = highest_priority_p;
      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;
      swtch(&(c->scheduler), p->context);
      c->proc = 0;
  }
  release(&ptable.lock);

 }
}
```

Explanation:
Modified the scheduler to always pick the runnable process with the highest priority (lowest integer value).

Step 4: Implement `setpriority()` System Call

File: `sysproc.c`
Code Added:

```
int sys_setpriority(void)
{
 struct proc *p = myproc();
 int priority;
 if (argint(0, &priority) < 0)
 {
   return -1;
 }
```

```
 p->priority = priority;
 return 0;
}
```

Explanation:
This system call allows a process to set its own static priority.

Step 5: Update System Call Definitions

Files: `syscall.h` and `syscall.c`

In `syscall.h`:

```
#define SYS_setpriority 30
```

In `syscall.c`:

```
extern int sys_setpriority(void);
```

```
[SYS_setpriority] sys_setpriority,
```

Explanation:
Declared and mapped the new system call so that the kernel recognizes `setpriority()`.

Step 6: User-Space Interface

Files: `user.h` and `usys.S`

In `user.h`:

```
int setpriority(int);
```

In `usys.S`:

```
SYSCALL(setpriority)
```

Explanation:
Made the new system call accessible from user programs.

Step 7: Create User-Space Test Program

File: `prioritytest.c`
Code:

```c
#include "types.h"
#include "stat.h"
#include "user.h"

int main() {
   int pids[5];
   int priorities[5] = {10, 20, 30, 40, 50};

   printf(1, "Starting priority scheduling test...\n");

   for (int i = 0; i < 5; i++) {
       if ((pids[i] = fork()) == 0) {
           setpriority(priorities[i]);
           printf(1, "Child %d with priority %d started.\n", getpid(),
priorities[i]);
           for (volatile int j = 0; j < 50000000; j++); // busy loop
           printf(1, "Child %d finished.\n", getpid());
           exit();
       }
   }

   for (int i = 0; i < 5; i++) wait();
   printf(1, "Priority scheduling test complete.\n");
   exit();
}
```

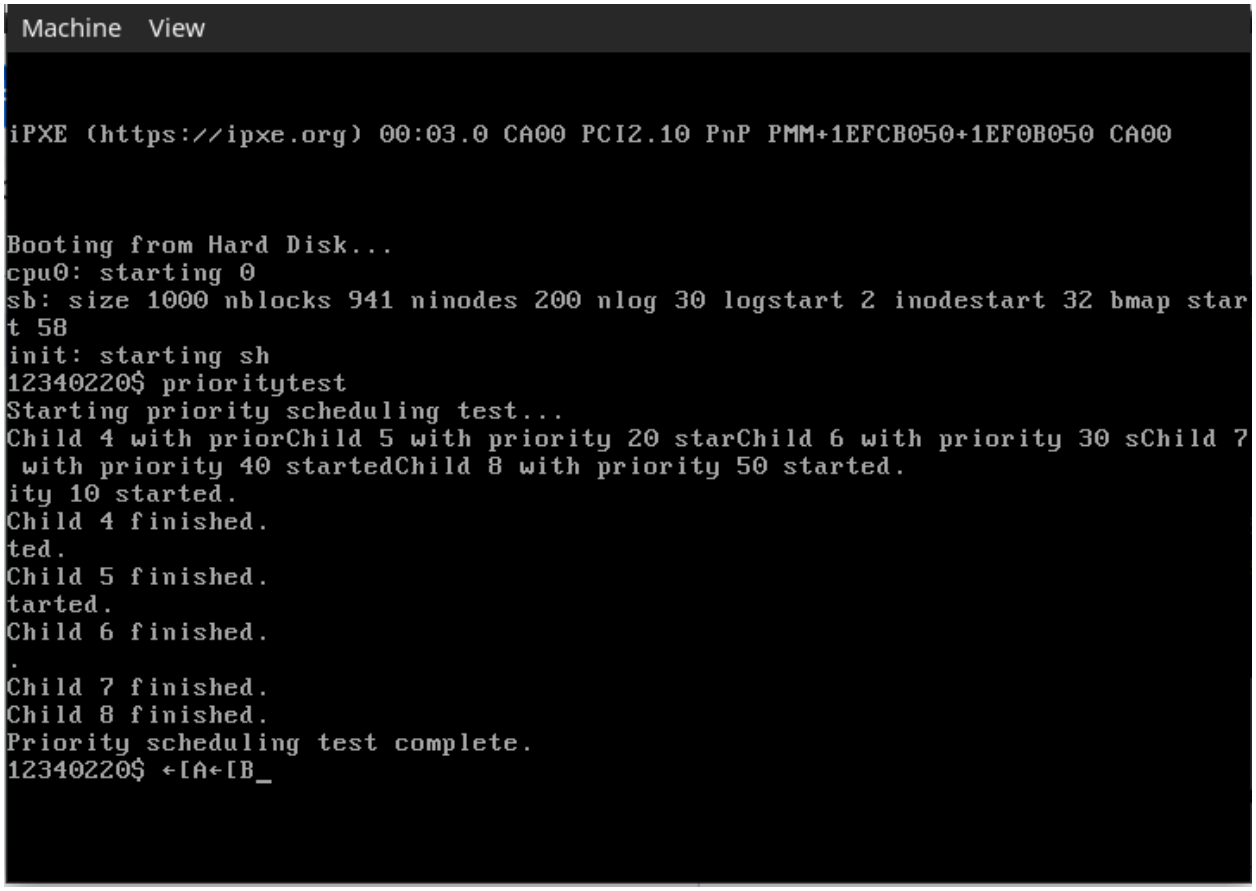
Explanation:
Created 5 child processes with different priorities. Busy loops simulate CPU work, demonstrating the scheduler respects static priorities.

Step 8: Build and Run

Changes to `Makefile`:
Added `_prioritytest` to `UPROGS`:

```
   _prioritytest\
```

Screenshot:


Machine View
iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCB050+1EF0B050 CA00
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
12340220$ prioritytest
Starting priority scheduling test...
Child 4 with priorChild 5 with priority 20 starChild 6 with priority 30 sChild 7
with priority 40 startedChild 8 with priority 50 started.
ity 10 started.
Child 4 finished.
ted.
Child 5 finished.
tarted.
Child 6 finished.
.
Child 7 finished.
Child 8 finished.
Priority scheduling test complete.
12340220$ ←[A←[B_