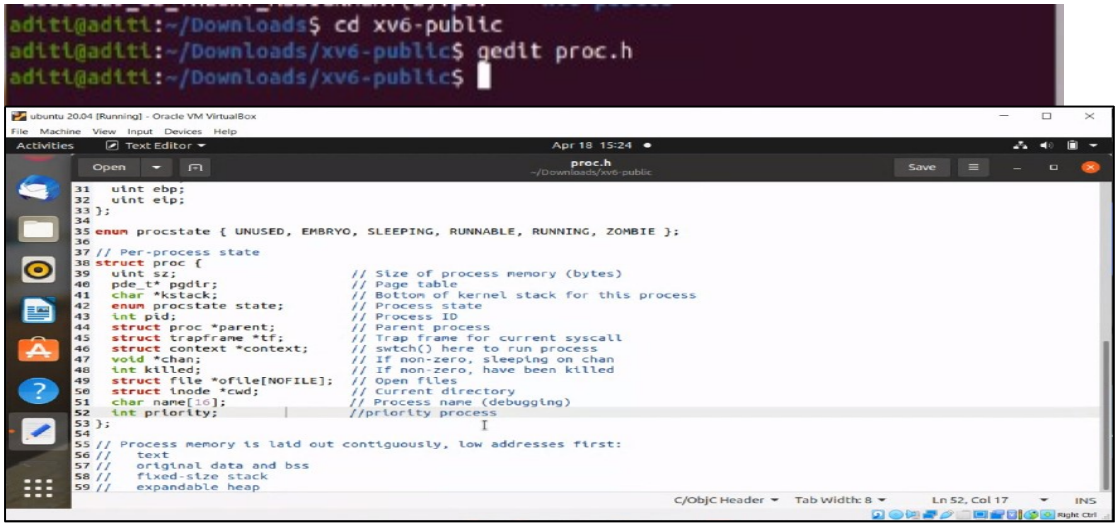
Question:-

Implement Round Robin and Priority Scheduling technique both in the single scheduler of xv6. The default scheduler of xv6 is round-robin. The implementation of priority scheduling link is as given below(You may follow other links). You go through both the scheduler and try to implement by combining both in one.

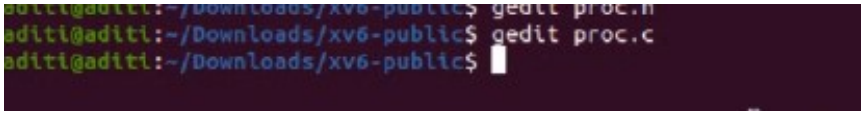
>> Create two system call ps and chpr for displaying processes context at that time and for changing priority respectively.

STEPS FOR SETTING UP SYSTEM CALL:-

Step-1:



Step-2:



Open ▾ | proc.c ~/Downloads/xv6-public | Save

```
79  acquire(&ptable.lock);
80
81  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
82    if(p->state == UNUSED)
83      goto found;
84
85  release(&ptable.lock);
86  return 0;
87
88 found:
89  p->state = EMBRYO;
90  p->pid = nextpid++;
91  p->priority = 10;  //default
92
93  release(&ptable.lock);
94
95  // Allocate kernel stack.
96  if((p->kstack = kalloc()) == 0){
97    p->state = UNUSED;
98    return 0;
99  }
00  sp = p->kstack + KSTACKSIZE;
01
02  // Leave room for trap frame.
03  sp -= sizeof *p->tf;
04  p->tf = (struct trapframe*)sp;
05
06  // Set up new context to start executing at forkret,
07  // which returns to trapret.
```

C ▾ Tab Width: 8 ▾ Ln 79, Col 25 ▾ INS

Step-3:

```
aditi@aditi:~/Downloads/xv6-public$ gedit exec.c
```

Open ▾ | exec.c ~/Downloads/xv6-public

```
76    if(copyout(pgdir, sp, argv[argc], strlen(argv[argc]) + 1) < 0)
77      goto bad;
78    ustack[3+argc] = sp;
79  }
80  ustack[3+argc] = 0;
81
82  ustack[0] = 0xffffffff;  // fake return PC
83  ustack[1] = argc;
84  ustack[2] = sp - (argc+1)*4;  // argv pointer
85
86  sp -= (3+argc+1) * 4;
87  if(copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)
88    goto bad;
89
90  // Save program name for debugging.
91  for(last=s=path; *s; s++)
92    if(*s == '/')
93      last = s+1;
94  safestrcpy(curproc->name, last, sizeof(curproc->name));
95
96  // Commit to the user image.
97  oldpgdir = curproc->pgdir;
98  curproc->pgdir = pgdir;
99  curproc->sz = sz;
100 curproc->tf->eip = elf.entry;  // main
101 curproc->tf->esp = sp;
102 curproc->priority = 3;
103 switchuvm(curproc);
104 freevm(oldpgdir);
105 return 0;
106
```

Step-4:

```
aditi@aditi:~/Downloads/xv6-public$ gedit syscall.h
```

Open ▾ syscall.h ~/Downloads/xv6-public

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_ps     22
#define SYS_chpr   23
```

Step-5:

```
aditi@aditi:~/Downloads/xv6-public$ gedit syscall.c
```

Open ▾ syscall.c ~/Downloads/xv6-public

```
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_ps(void);
extern int sys_chpr(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
```

Open ▾ syscall.c ~/Downloads/xv6-public

```
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_ps]      sys_ps,
[SYS_chpr]    sys_chpr,
};
```

Step-6:

```
aditi@aditi:~/Downloads/xv6-public$ gedit defs.h
```

Open — defs.h — ~/Downloads/xv6-public

```
115 void            scheduler(void) __attribute__((noreturn));
116 void            sched(void);
117 void            setproc(struct proc*);
118 void            sleep(void*, struct spinlock*);
119 void            userinit(void);
120 int             wait(void);
121 void            wakeup(void*);
122 void            yield(void);
123 int             ps(void);
124 int             chpr(int pid, int priority);
125
126 // swtch.S
127 void            swtch(struct context**, struct context*);
128
129 // spinlock.c
130 void            acquire(struct spinlock*);
131 void            getcallerpcs(void*, uint*);
132 int             holding(struct spinlock*);
133 void            initlock(struct spinlock*, char*);
134 void            release(struct spinlock*);
135 void            pushcli(void);
136 void            popcli(void);
```

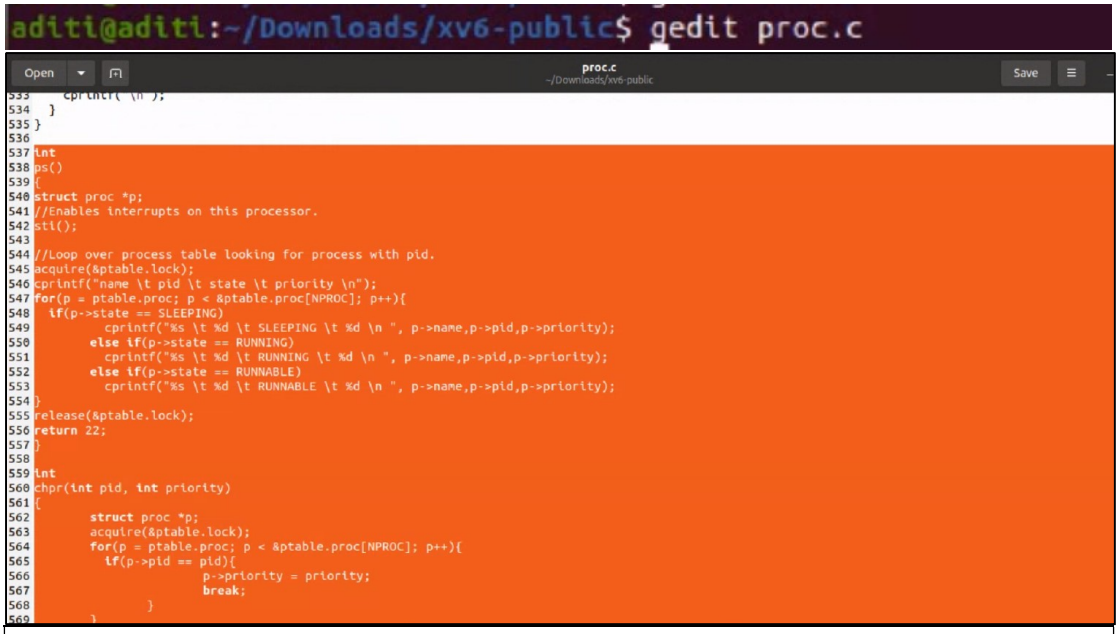
Step-7:

```
aditi@aditi:~/Downloads/xv6-public$ gedit user.h
```

Open — user.h — ~/Downloads/xv6-public

```
1 struct stat;
2 struct rtcdate;
3
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int ps(void);
27 int chpr(int pid, int priority);
28
```

Step-8:




```
int
ps()
{
struct proc *p;
//Enables interrupts on this processor.
sti();

//Loop over process table looking for process with pid.
acquire(&ptable.lock);
cprintf("name \t pid \t state \t priority \n");
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
 if(p->state == SLEEPING)
            cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name,p->pid,p->priority);
          else if(p->state == RUNNING)
            cprintf("%s \t %d \t RUNNING \t %d \n ", p->name,p->pid,p->priority);
          else if(p->state == RUNNABLE)
            cprintf("%s \t %d \t RUNNABLE \t %d \n ", p->name,p->pid,p->priority);
}
release(&ptable.lock);
return 22;
}

int
chpr(int pid, int priority)
{
          struct proc *p;
          acquire(&ptable.lock);
          for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
           if(p->pid == pid){
                              p->priority = priority;
                              break;
                    }
          }
          release(&ptable.lock);
          return pid;
}
```

Step-9:

```
aditi@aditi:~/Downloads/xv6-public$ gedit sysproc.c
```

Open ▾ | *sysproc.c ~/Downloads/xv6-public

```
78 }
79
80 // return how many clock tick interrupts have occurred
81 // since start.
82 int
83 sys_uptime(void)
84 {
85   uint xticks;
86
87   acquire(&tickslock);
88   xticks = ticks;
89   release(&tickslock);
90   return xticks;
91 }
92
93
94 int
95 sys_ps()
96 {
97     return ps();
98 }
99
100 int
101 sys_chpr()
102 {
103    int pid,pr;
104    if(argint(0,&pid)<0)
105        return -1;
106    if(argint(1,&pr)<0)
107        return -1;
108
109    return chpr(pid, pr);
110 }
111
```

Step-10:

```
aditi@aditi:~/Downloads/xv6-public$ gedit usys.S
```

Open ▾ | usys.S ~/Downloads/xv6-public

```
1 #include "syscall.h"
2 #include "traps.h"
3
4 #define SYSCALL(name) \
5   .globl name; \
6   name: \
7     movl $SYS_ ## name, %eax; \
8     int $T_SYSCALL; \
9     ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(ps)
33 SYSCALL(chpr)
34
```

Step-11:

1.) ps.c

```
aditi@aditi:~/Downloads/xv6-public$ gedit ps.c
```

Open ps.c ~/Downloads/xv6-public

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

int main()
{
        ps();
        exit();
}
```

2.) pri.c

```
aditi@aditi:~/Downloads/xv6-public$ gedit pri.c
```

Open pri.c ~/Downloads/xv6-public

```
#include"types.h"
#include"stat.h"
#include"user.h"
#include"fcntl.h"
int main(int argc, char *argv[])
{
   int pri, pid;
   pid = atoi(argv[1]);
   pri = atoi(argv[2]);
   if(pri<0 || pri>20)
   {
       printf(2,"NOT VALID priority. (0-20)!\n");
       exit();
   }
   else
   {
      chpr(pid, pri); // calling the system call tochangethepriority
   }
   printf(1,"Priority of process with PID %d changed.\n",pid);
   exit();
}
```

Step-12:

The default scheduler of xv6 is based on Round Robin algorithm.

Now, we have to create a c program which creates a number of child process as mentioned by the user and consumes CPU time for testing our system calls and scheduling. So we create a new file *dpro.c(dummy program)*and write the following code:

```
aditi@aditi:~/Downloads/xv6-public$ gedit dpro.c
```

Open — dpro.c — ~/Downloads/xv6-public

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

int main(int argc, char *argv[]) {
  int pid;
  int k, n;
  int x, z;

  if(argc < 2)
        n = 1; //Default
  else
        n = atoi(argv[1]);
  if (n < 0 ||n > 20)
        n = 2;
  x = 0;
  pid = 0;

  for ( k = 0; k < n; k++ ) {
    pid = fork ();
    if ( pid < 0 ) {
      printf(1, "%d failed in fork!\n", getpid());
    } else if (pid > 0) {
      // parent
      printf(1, "Parent %d creating child %d\n",getpid(), pid);
      wait();
      }
      else{
        printf(1,"Child %d created\n",getpid());
        for(z = 0; z < 4000000000; z+=1)
            x = x + 3.14*89.64; //Useless calculation to consume CPU Time
        break;
      }
  }
  exit();
}
```

Step-13:

```
aditi@aditi:~/Downloads/xv6-public$ gedit Makefile
```

Open — Makefile — ~/Downloads/xv6-public

```
.PRECIOUS: %.o

UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\
        _ps\
        _pri\
        _dpro\

fs.img: mkfs README $(UPROGS)
        ./mkfs fs.img README $(UPROGS)

-include *.d

clean:
```

Open — Makefile — ~/Downloads/xv6-public

```
240         @echo "*** Now run 'gdb'." 1>&2
241         $(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)
242
243 qemu-nox-gdb: fs.img xv6.img .gdbinit
244         @echo "*** Now run 'gdb'." 1>&2
245         $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)
246
247 # CUT HERE
248 # prepare dist for students
249 # after running make dist, probably want to
250 # rename it to rev0 or rev1 or so on and then
251 # check in that version.
252
253 EXTRA=\
254         mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
255         ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c ps.c pri.c d.c\
256         printf.c umalloc.c\
257         README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
258         .gdbinit.tmpl gdbutil\
259
260 dist:
261         rm -rf dist
262         mkdir dist
263         for i in $(FILES); \
264         do \
265                 grep -v PAGEBREAK $$i >dist/$$i; \
266         done
267         sed '/CUT HERE/,$$d' Makefile >dist/Makefile
268         echo >dist/runoff.spec
269         cp $(EXTRA) dist
```

---Implemented the system calls successfully.

>> Running the whole configuration:-

We make the required changes in scheduler function in proc.c file.

```
aditi@aditi:~/Downloads/xv6-public$ gedit proc.c
```

Open — proc.c — ~/Downloads/xv6-public

```
323 void
324 scheduler(void)
325 {
326   struct proc *p, *p1;
327   struct cpu *c = mycpu();
328   c->proc = 0;
329
330   for(;;){
331     // Enable interrupts on this processor.
332     sti();
333     struct proc *highP;
334     // Loop over process table looking for process to run.
335     acquire(&ptable.lock);
336     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
337       if(p->state != RUNNABLE)
338         continue;
339
340       // Switch to chosen process.  It is the process's job
341       // to release ptable.lock and then reacquire it
342       // before jumping back to us.
343       highP = p;
344       //choose one with highest priority
345       for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){
346         if(p1->state != RUNNABLE)
347           continue;
348         if(highP->priority > p1->priority)   //larger value, lower priority
349           highP = p1;
350       }
351       p = highP;
352       c->proc = p;
353       switchuvm(p);
354       p->state = RUNNING;
355
356       swtch(&(c->scheduler), p->context);
357       switchkvm();
358
```

```
Open                                                    proc.c
                                                        ~/Downloads/xv6-publ
339
340       // Switch to chosen process.  It is the process's job
341       // to release ptable.lock and then reacquire it
342       // before jumping back to us.
343       highP = p;
344       //choose one with highest priority
345       for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){
346         if(p1->state != RUNNABLE)
347           continue;
348         if(highP->priority > p1->priority)   //larger value, lower priority
349           highP = p1;
350       }
351       p = highP;
352       c->proc = p;
353       switchuvm(p);
354       p->state = RUNNING;
355
356       swtch(&(c->scheduler), p->context);
357       switchkvm();
358
359       // Process is done running for now.
360       // It should have changed its p->state before coming back.
361       c->proc = 0;
362     }
363     release(&ptable.lock);
364
365   }
366
```

We have implemented the system calls and changed the scheduling policy in xv6.

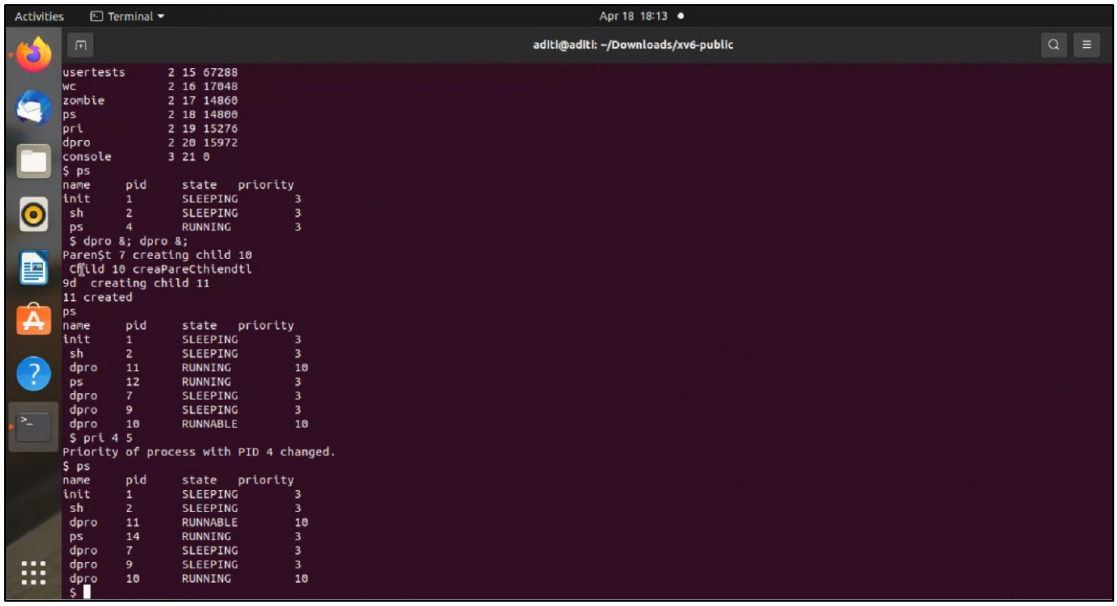
We have called dpro functions twice simultaneously.

<u>OUTPUT:--</u>

Make qemu-nox

```
Activities   Terminal                                  Apr 18 18:11
                                                       aditi@aditi: ~/Downloads/xv6-public
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00


Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 16312
echo           2 4 15168
forktest       2 5 9472
grep           2 6 18528
init           2 7 15748
kill           2 8 15196
ln             2 9 15048
ls             2 10 17680
mkdir          2 11 15292
rm             2 12 15272
sh             2 13 27904
stressfs       2 14 16184
usertests      2 15 67288
wc             2 16 17048
zombie         2 17 14860
ps             2 18 14800
pri            2 19 15276
dpro           2 20 15972
console        3 21 0
$ ps
name     pid     state   priority
init     1       SLEEPING        3
 sh      2       SLEEPING        3
 ps      4       RUNNING         3
$
```


Activities
Terminal
Apr 18 18:13
aditi@aditi: ~/Downloads/xv6-public
usertests 2 15 67288
wc 2 16 17048
zombie 2 17 14860
ps 2 18 14800
pri 2 19 15276
dpro 2 20 15972
console 3 21 0
$ ps
name pid state priority
init 1 SLEEPING 3
sh 2 SLEEPING 3
ps 4 RUNNING 3
$ dpro &; dpro &;
Paren$t 7 creating child 10
Cfild 10 creaPareCthiendtl
9d creating child 11
11 created
ps
name pid state priority
init 1 SLEEPING 3
sh 2 SLEEPING 3
dpro 11 RUNNING 10
ps 12 RUNNING 3
dpro 7 SLEEPING 3
dpro 9 SLEEPING 3
dpro 10 RUNNABLE 10
$ pri 4 5
Priority of process with PID 4 changed.
$ ps
name pid state priority
init 1 SLEEPING 3
sh 2 SLEEPING 3
dpro 11 RUNNABLE 10
ps 14 RUNNING 3
dpro 7 SLEEPING 3
dpro 9 SLEEPING 3
dpro 10 RUNNING 10
$