Open ▾ proc.h ~/Desktop/xv6-public Save

```
  uint ebp;
  uint eip;
};

enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int priority;                // Process Priority
};

// Process memory is laid out contiguously, low addresses first:
//   text
//   original data and bss
//   fixed-size stack
```

C/ObjC Header ▾ Tab Width: 8 ▾ Ln 52, Col 51 ▾ INS

Open ▾ proc.c ~/Desktop/xv6-public Save

```
    if(p->state == UNUSED)
      goto found;

  release(&ptable.lock);
  return 0;

found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->priority = 10;    //default priority
  release(&ptable.lock);

  // Allocate kernel stack.
  if((p->kstack = kalloc()) == 0){
    p->state = UNUSED;
    return 0;
  }
  sp = p->kstack + KSTACKSIZE;

  // Leave room for trap frame.
  sp -= sizeof *p->tf;
  p->tf = (struct trapframe*)sp;

  // Set up new context to start executing at forkret,
  // which returns to trapret.
  sp -= 4;
  *(uint*)sp = (uint)trapret;
```

C ▾ Tab Width: 8 ▾ Ln 92, Col 25 ▾ INS

Open | proc.c ~/Desktop/xv6-public | Save

```
//current process states
int
cps()
{
        struct proc *p;
        //Enable interrupts on this processor.
        sti();

        //Loop over the 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;
}
```

C | Tab Width: 8 | Ln 553, Col 94 | INS

Open | foo.c ~/Desktop/xv6-public | Save

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

int
main(int argc, char *argv[])
{
  int  k, n, id;
  double x = 0,  z;

  if(argc < 2 )
    n = 1;       //default value
  else
    n = atoi ( argv[1] ); //from command line
  if ( n < 0 || n > 20 )
    n = 2;
  x = 0;
  id = 0;
  for ( k = 0; k < n; k++ ) {
    id = fork ();
    if ( id < 0 ) {
      printf(1, "%d failed in fork!\n", getpid() );
    } else if ( id > 0 ) {  //parent
      printf(1, "Parent %d creating child  %d\n", getpid(), id );
      wait ();
   } else {   // child
      printf(1, "Child %d created\n",getpid() );
```

C | Tab Width: 8 | Ln 17, Col 11 | INS

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

```
>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;
}
//change priority
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;
}
```

C ▾ | Tab Width: 8 ▾ | Ln 572, Col 4 | INS

Open ▾ | sysproc.c | ~/Desktop/xv6-public | Save

```
  return xticks;
}

int
sys_cps(void)
{
  return cps();
}

int
sys_getppid(void)
{
        return myproc()->parent->pid;
}
int
sys_chpr (void)
{
  int pid, pr;
  if(argint(0, &pid) < 0)
    return -1;
  if(argint(1, &pr) < 0)
    return -1;

  return chpr ( pid, pr );
}
```

C ▾ | Tab Width: 8 ▾ | Ln 107, Col 15 | INS

Open ▾ | syscall.h ~/Desktop/xv6-public | Save

```
// 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_cps    22
#define SYS_getppid 23
#define SYS_chpr   24
```

C/ObjC Header ▾ Tab Width: 8 ▾ Ln 25, Col 22 ▾ INS

Open ▾ | user.h ~/Desktop/xv6-public | Save

```
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int cps(void);
int getppid(void);
int chpr(void);

// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void printf(int, const char*, ...);
char* gets(char*, int max);
```

Open ▾ | usys.S ~/Desktop/xv6-public | Save

```
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(cps)
SYSCALL(getppid)
SYSCALL(chpr)
```

C ▾ Tab Width: 8 ▾ Ln 34, Col 14 ▾ INS

Open ▾ | syscall.c ~/Desktop/xv6-public | Save

```
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_cps(void);
extern int sys_getppid(void);
extern int sys_chpr(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec
```

C ▾ Tab Width: 8 ▾ Ln 113, Col 24 ▾ INS

Open | syclall.c | ~/Desktop/xv6-public | Save

```
[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_cps]     sys_cps,
[SYS_getppid] sys_getppid,
[SYS_chpr]    sys_chpr,
};

void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
```

C | Tab Width: 8 | Ln 134, Col 24 | INS

Open | Makefile | ~/Desktop/xv6-public | Save

```
UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _ps\
        _parent\
        _nice\
        _zombie\

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

-include *.d

clean:
        rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
        *.o *.d *.asm *.sym vectors.S bootblock entryother \
        initcode initcode.out kernel xv6.img fs.img kernelmemfs \
```

Makefile | Tab Width: 8 | Ln 190, Col 1 | INS

Open | Makefile ~/Desktop/xv6-public | Save

```
	$(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)

qemu-nox-gdb: fs.img xv6.img .gdbinit
	@echo "*** Now run 'gdb'." 1>&2
	$(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)

# CUT HERE
# prepare dist for students
# after running make dist, probably want to
# rename it to rev0 or rev1 or so on and then
# check in that version.

EXTRA=\
	mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
	ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c ps.c parent.c nice.c
zombie.c\
	printf.c umalloc.c\
	README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
	.gdbinit.tmpl gdbutil\

dist:
	rm -rf dist
	mkdir dist
	for i in $(FILES); \
	do \
		grep -v PAGEBREAK $$i >dist/$$i; \
	done
	sed '/CUT HERE/,$$d' Makefile >dist/Makefile
```

Makefile | Tab Width: 8 | Ln 255, Col 80 | INS

Open | nice.c ~/Desktop/xv6-public | Save

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

int
main(int argc, char *argv[])
{
  int priority, pid;

  if(argc < 3 ){
      printf(2, "Usage: nice pid priority\n" );
      exit();
  }
  pid = atoi ( argv[1] );
  priority = atoi ( argv[2] );
  if ( priority < 0 || priority > 20 ) {
      printf(2, "Invalid priority (0-20)!\n" );
      exit();
  }
  chpr ( pid, priority );

  exit();
}
```

C | Tab Width: 8 | Ln 25, Col 1 | INS