Assignment 1

Group no -26

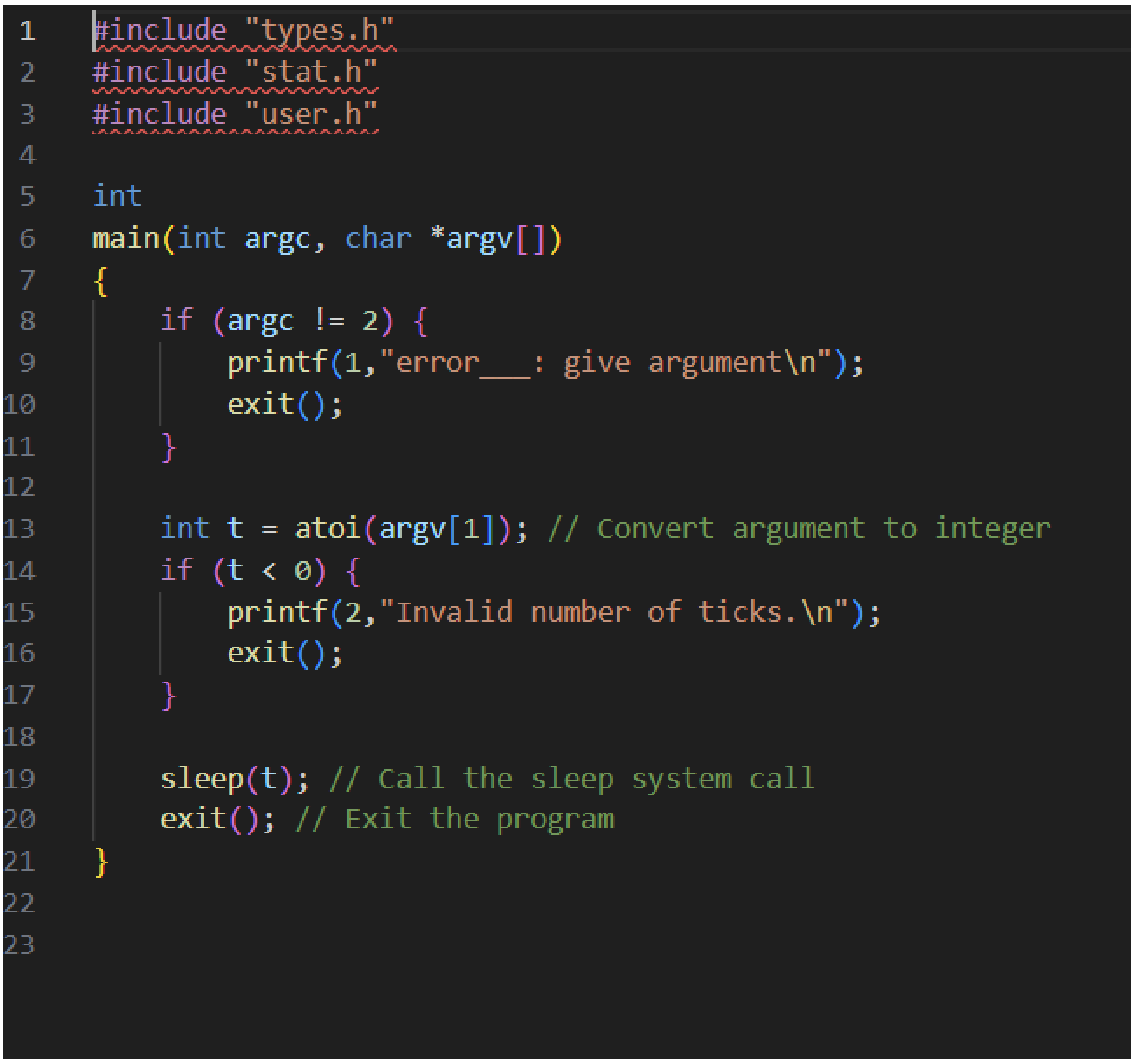
220101078 Pentakota Rama Vardhan

220101079 Pihul Lalotra

220101083 Rahul

220101088 Rohan Kumar Mahto

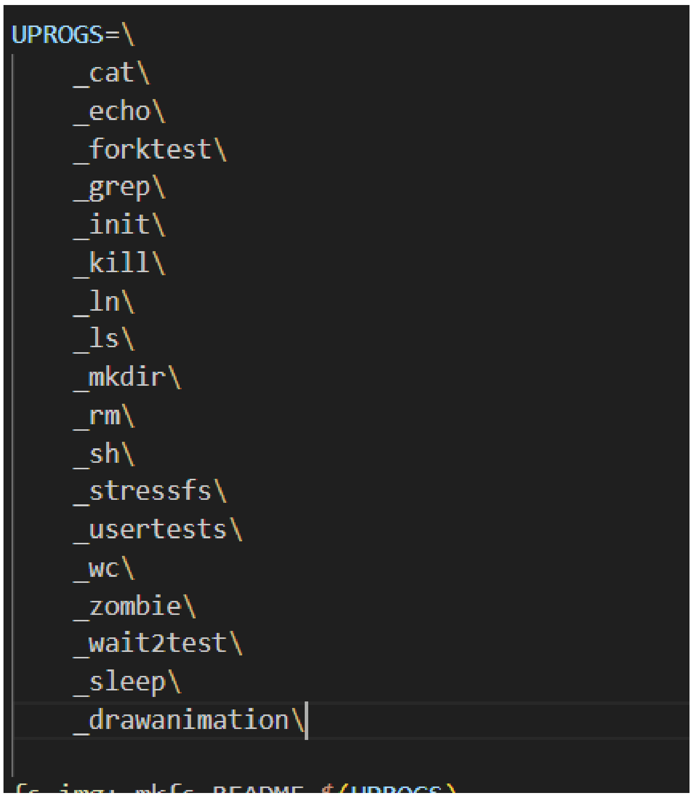
Task 1.1: Sleep



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

int
main(int argc, char *argv[])
{
    if (argc != 2) {
        printf(1,"error___: give argument\n");
        exit();
    }

    int t = atoi(argv[1]); // Convert argument to integer
    if (t < 0) {
        printf(2,"Invalid number of ticks.\n");
        exit();
    }

    sleep(t); // Call the sleep system call
    exit(); // Exit the program
}
```

Code Breakdown

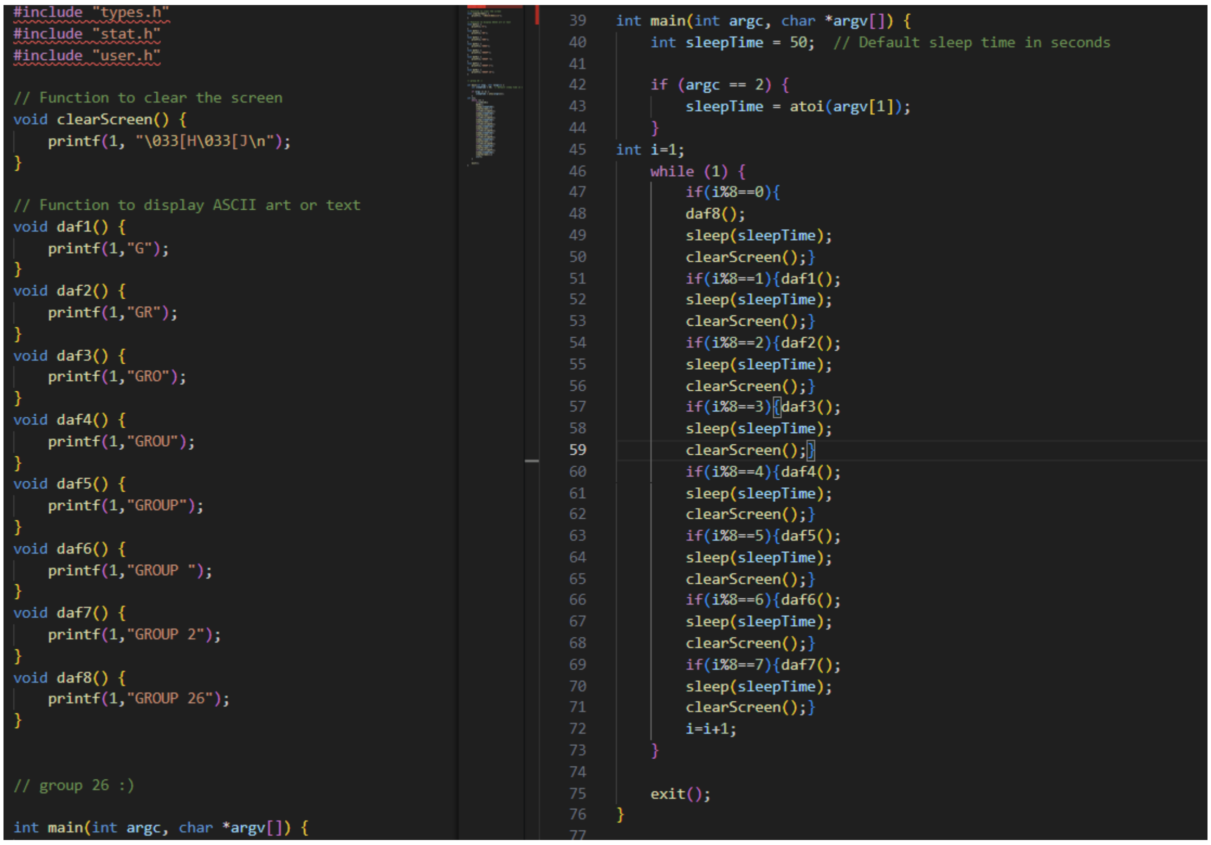
1. **Header Inclusion:**
   - **types.h**: Provides basic data types like int.
   - **stat.h**: Defines structures for file and directory information.
   - **user.h**: Contains system calls and other user-level functions.
2. **Main Function:**
   - **Argument Parsing:** Checks if the correct number of arguments is provided (exactly one) and ensures the argument is a positive integer.
   - **Conversion:** Converts the string argument to an integer representing the number of ticks to sleep.
   - **System Call:** Calls the sleep system call, passing the specified number of ticks.
   - **Exit:** Terminates the program.



```
UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\
	_wait2test\
	_sleep\
	_drawanimation\
```

Inside makefile we added sleep program to UPROGS.

Task 1.2: User Program to display animation



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

// Function to clear the screen
void clearScreen() {
    printf(1, "\033[H\033[J\n");
}

// Function to display ASCII art or text
void daf1() {
    printf(1,"G");
}
void daf2() {
    printf(1,"GR");
}
void daf3() {
    printf(1,"GRO");
}
void daf4() {
    printf(1,"GROU");
}
void daf5() {
    printf(1,"GROUP");
}
void daf6() {
    printf(1,"GROUP ");
}
void daf7() {
    printf(1,"GROUP 2");
}
void daf8() {
    printf(1,"GROUP 26");
}


// group 26 :)

int main(int argc, char *argv[]) {
```

```c
int main(int argc, char *argv[]) {
    int sleepTime = 50;  // Default sleep time in seconds

    if (argc == 2) {
        sleepTime = atoi(argv[1]);
    }
int i=1;
    while (1) {
        if(i%8==0){
        daf8();
        sleep(sleepTime);
        clearScreen();}
        if(i%8==1){daf1();
        sleep(sleepTime);
        clearScreen();}
        if(i%8==2){daf2();
        sleep(sleepTime);
        clearScreen();}
        if(i%8==3){daf3();
        sleep(sleepTime);
        clearScreen();}
        if(i%8==4){daf4();
        sleep(sleepTime);
        clearScreen();}
        if(i%8==5){daf5();
        sleep(sleepTime);
        clearScreen();}
        if(i%8==6){daf6();
        sleep(sleepTime);
        clearScreen();}
        if(i%8==7){daf7();
        sleep(sleepTime);
        clearScreen();}
        i=i+1;
    }

    exit();
}
```

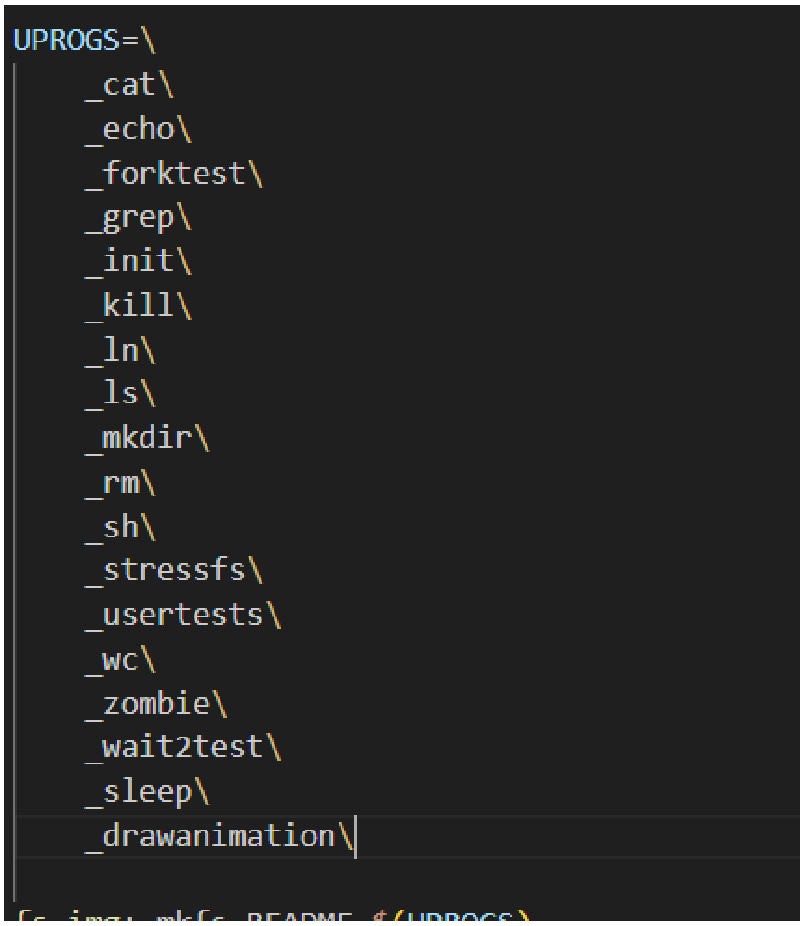
Purpose

We implemented an animation program that displays a sequence of ASCII characters, simulating a scrolling effect. The animation consists of eight different frames, each representing a different stage of the sequence.

Functionality Breakdown

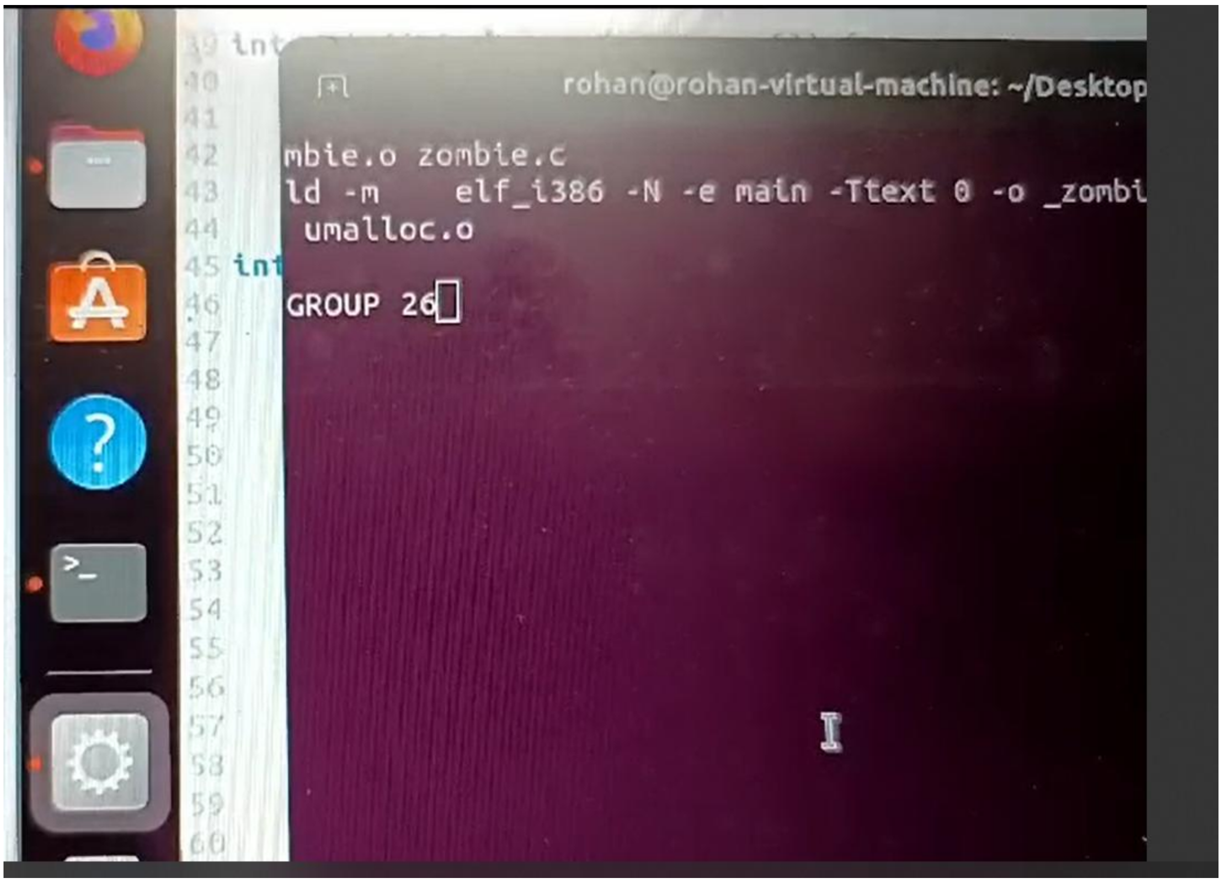
1. **Header Inclusion:**
    - types.h, stat.h, and user.h are included to provide basic data types, file system operations, and system calls, respectively.
2. **Function Declarations:**
    - clearScreen(): Clears the terminal screen.
    - daf1(), daf2(), ..., daf8(): Functions to display individual frames of the animation.

3. **Main Function:**
   - **Command-Line Argument:** Checks if a command-line argument is provided. If so, it sets the sleepTime variable to the specified value. Otherwise, the default sleep time of 50 seconds is used.
   - **Infinite Loop:**
     - Increments the i variable in each iteration.
     - Uses the modulo operator (%) to determine which frame to display based on the value of i.
     - Calls the corresponding daf function to display the frame.
     - Pauses execution for sleepTime seconds using the sleep system call.
     - Clears the screen using the clearScreen() function.
4. **Frame Display Functions:**
   - Each daf function prints a specific sequence of characters to represent a frame of the animation.



```
UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\
	_wait2test\
	_sleep\
	_drawanimation\

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

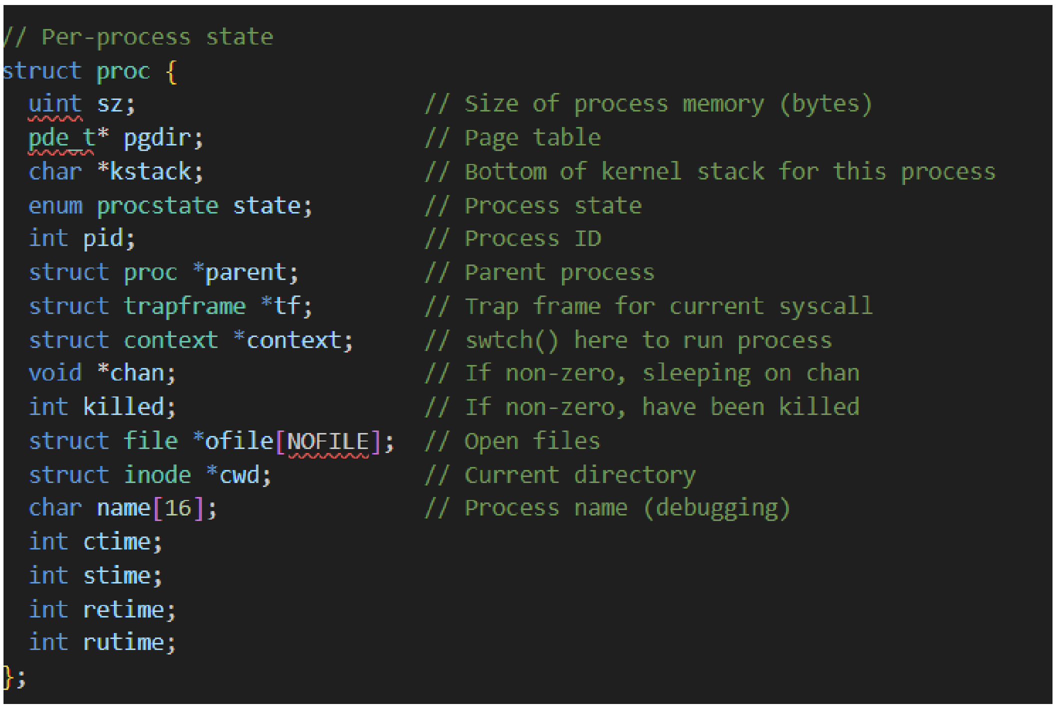
Inside makefile we added drawanimation program to UPROGS.



rohan@rohan-virtual-machine: ~/Desktop

```
mbie.o zombie.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _zombi
 umalloc.o

GROUP 26
```

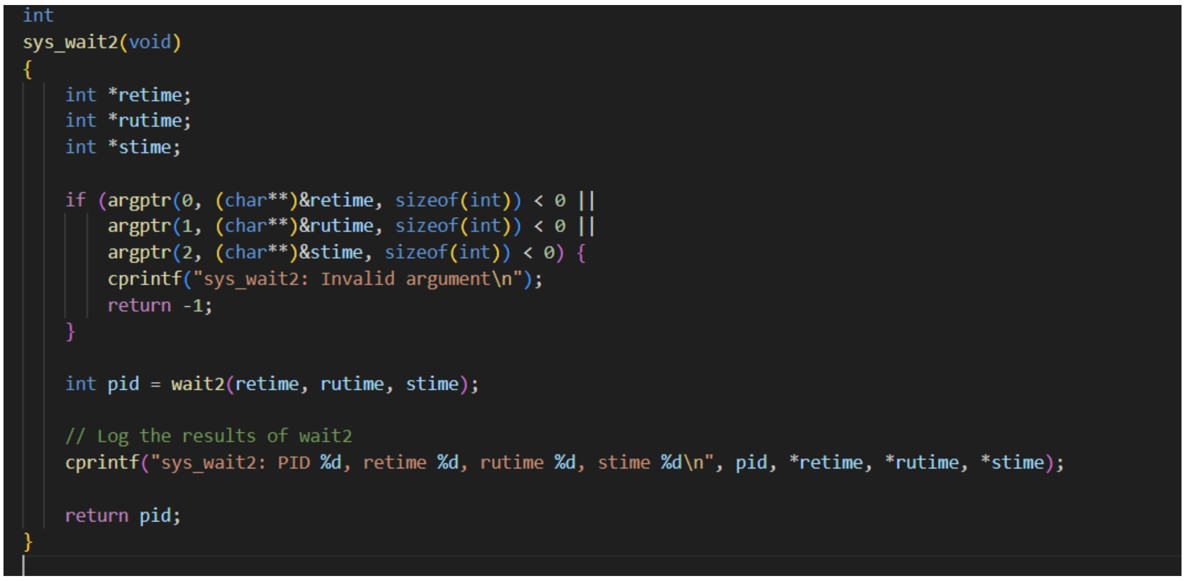
Task 1.3: Statistics



```
// 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 ctime;
  int stime;
  int retime;
  int rutime;
};
```

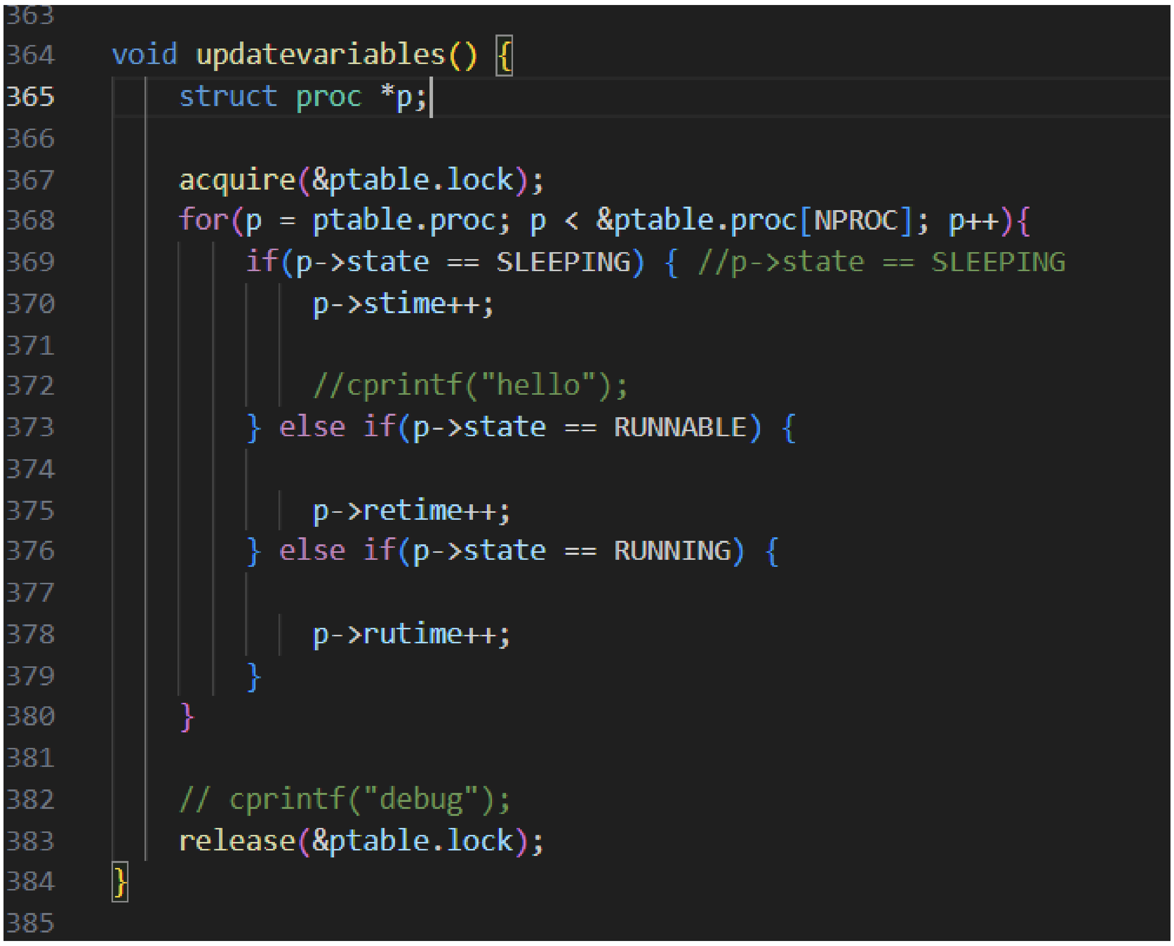
We added four new fields to the proc structure:

- ctime: Creation time of the process
- stime: Time spent in the sleeping state
- retime: Time spent in the ready or running state
- rutime: Time spent in the running state



```c
int
sys_wait2(void)
{
    int *retime;
    int *rutime;
    int *stime;

    if (argptr(0, (char**)&retime, sizeof(int)) < 0 ||
        argptr(1, (char**)&rutime, sizeof(int)) < 0 ||
        argptr(2, (char**)&stime, sizeof(int)) < 0) {
        cprintf("sys_wait2: Invalid argument\n");
        return -1;
    }

    int pid = wait2(retime, rutime, stime);

    // Log the results of wait2
    cprintf("sys_wait2: PID %d, retime %d, rutime %d, stime %d\n", pid, *retime, *rutime, *stime);

    return pid;
}
```

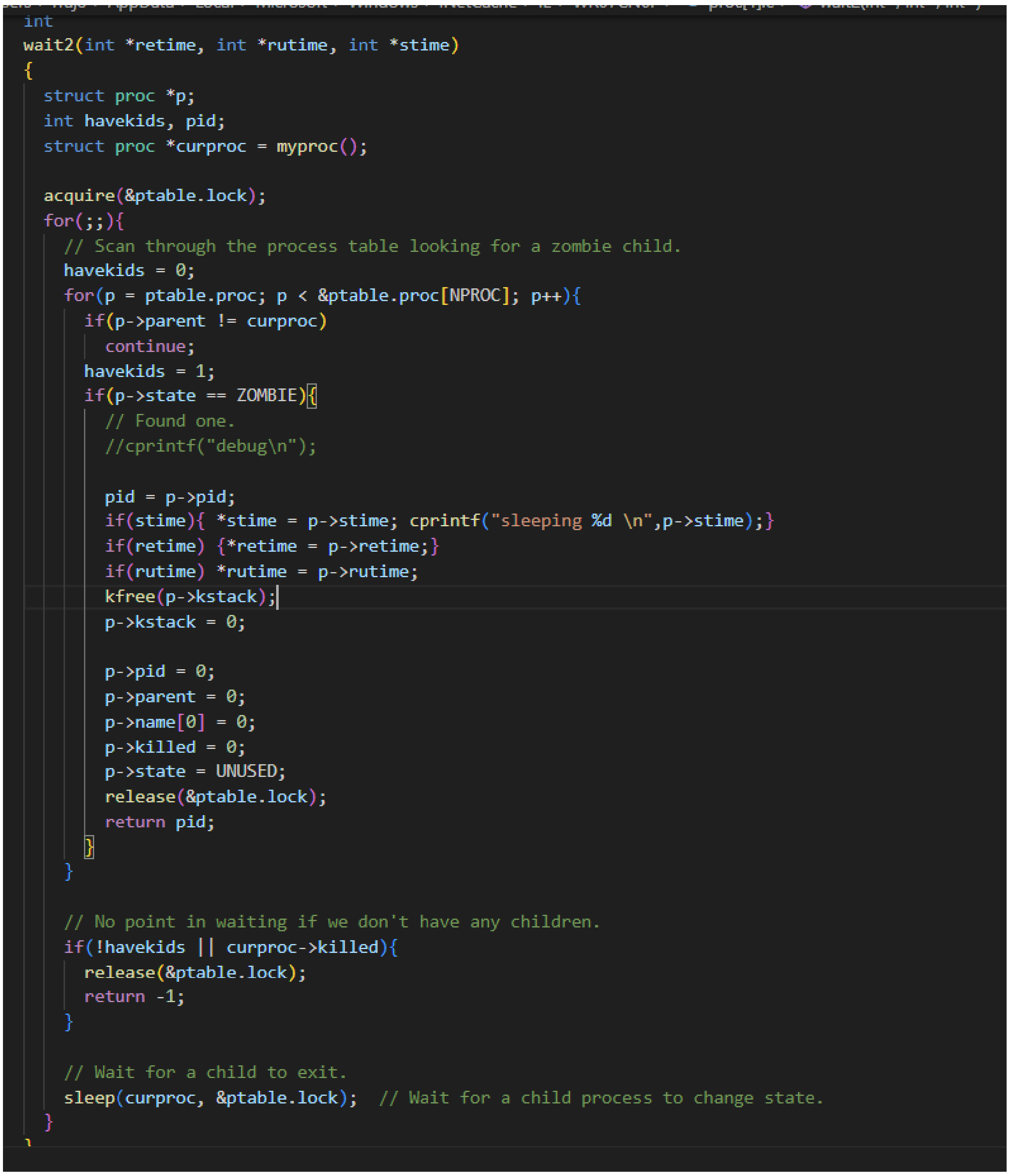
Inside sysproc.c we added this



```c

void updatevariables() {
    struct proc *p;

    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
        if(p->state == SLEEPING) { //p->state == SLEEPING
            p->stime++;

            //cprintf("hello");
        } else if(p->state == RUNNABLE) {

            p->retime++;
        } else if(p->state == RUNNING) {

            p->rutime++;
        }
    }

    // cprintf("debug");
    release(&ptable.lock);
}

```

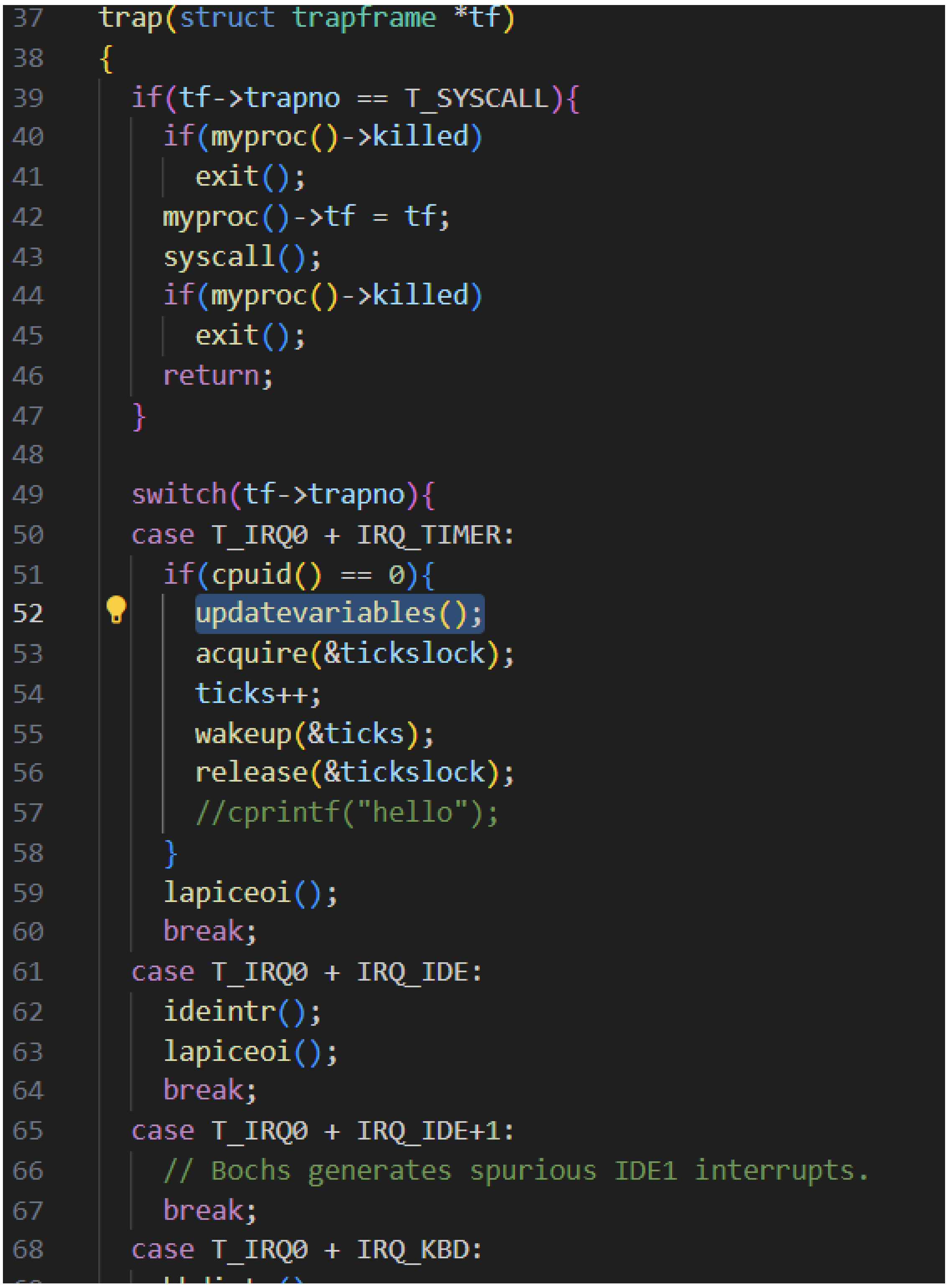
We added udatevariables() in proc.c

- The wait2 system call takes pointers to three integers as arguments: retime, rutime, and stime.
- It waits for a child process to terminate.
- Upon termination, it assigns the accumulated values of retime, rutime, and stime for the terminated process to the corresponding pointers.
- It returns the PID of the terminated child process or -1 if an error occurs.



```c
int
wait2(int *retime, int *rutime, int *stime)
{
  struct proc *p;
  int havekids, pid;
  struct proc *curproc = myproc();

  acquire(&ptable.lock);
  for(;;){
    // Scan through the process table looking for a zombie child.
    havekids = 0;
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->parent != curproc)
        continue;
      havekids = 1;
      if(p->state == ZOMBIE){
        // Found one.
        //cprintf("debug\n");

        pid = p->pid;
        if(stime){ *stime = p->stime; cprintf("sleeping %d \n",p->stime);}
        if(retime) {*retime = p->retime;}
        if(rutime) *rutime = p->rutime;
        kfree(p->kstack);
        p->kstack = 0;

        p->pid = 0;
        p->parent = 0;
        p->name[0] = 0;
        p->killed = 0;
        p->state = UNUSED;
        release(&ptable.lock);
        return pid;
      }
    }

    // No point in waiting if we don't have any children.
    if(!havekids || curproc->killed){
      release(&ptable.lock);
      return -1;
    }

    // Wait for a child to exit.
    sleep(curproc, &ptable.lock);  // Wait for a child process to change state.
  }
}
```

Inside proc.c we added this



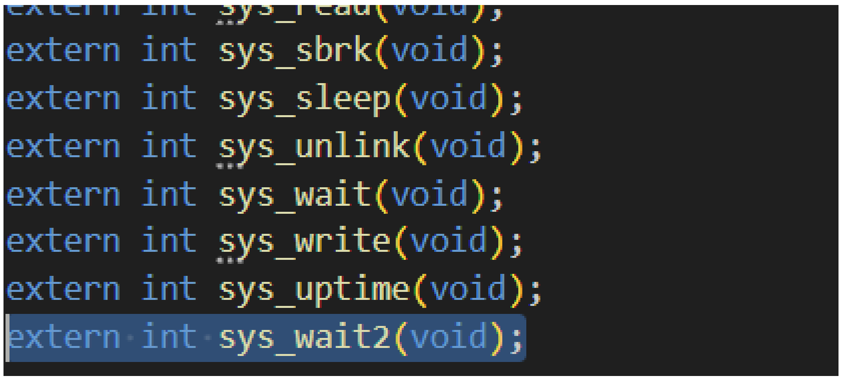
```c
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(myproc()->killed)
      exit();
    myproc()->tf = tf;
    syscall();
    if(myproc()->killed)
      exit();
    return;
  }

  switch(tf->trapno){
  case T_IRQ0 + IRQ_TIMER:
    if(cpuid() == 0){
      updatevariables();
      acquire(&tickslock);
      ticks++;
      wakeup(&ticks);
      release(&tickslock);
      //cprintf("hello");
    }
    lapiceoi();
    break;
  case T_IRQ0 + IRQ_IDE:
    ideintr();
    lapiceoi();
    break;
  case T_IRQ0 + IRQ_IDE+1:
    // Bochs generates spurious IDE1 interrupts.
    break;
  case T_IRQ0 + IRQ_KBD:
```

We called updatevariables() in trap.c so that

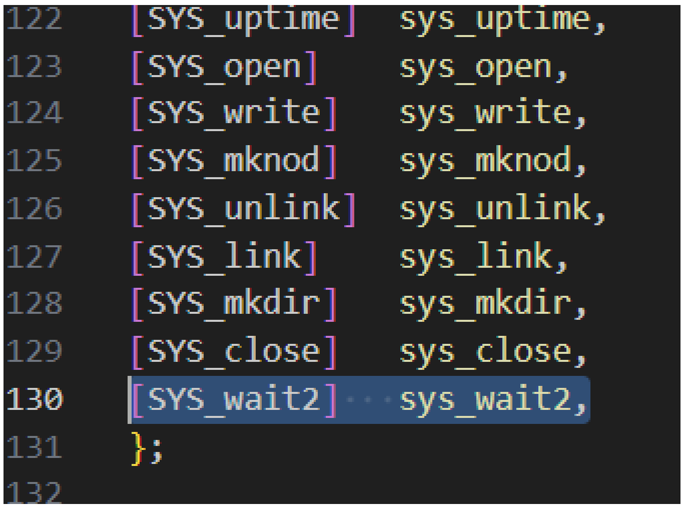
retime

stime

rutime will get updated through this function.

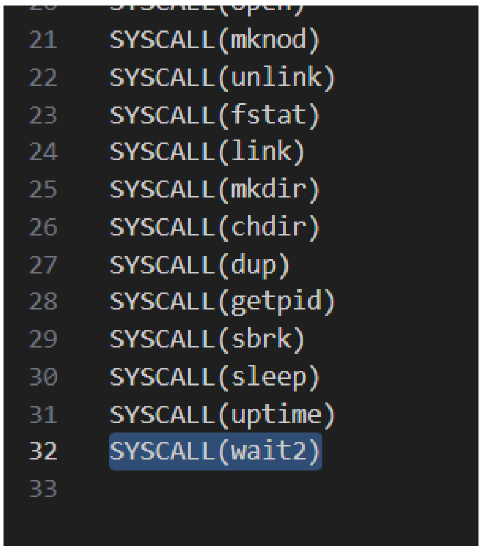


```
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_wait2(void);
```



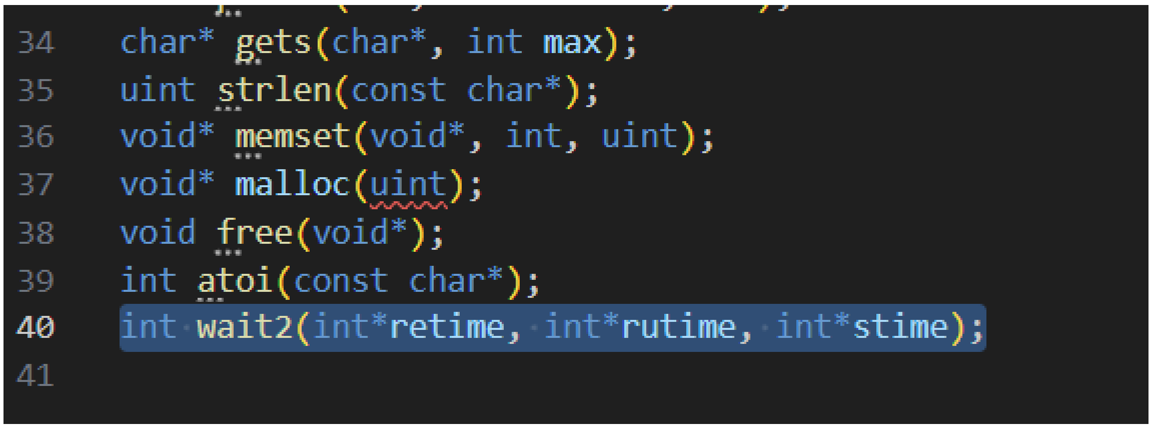
```
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_wait2]   sys_wait2,
};
```

We added sys_wait2 in syscall.c



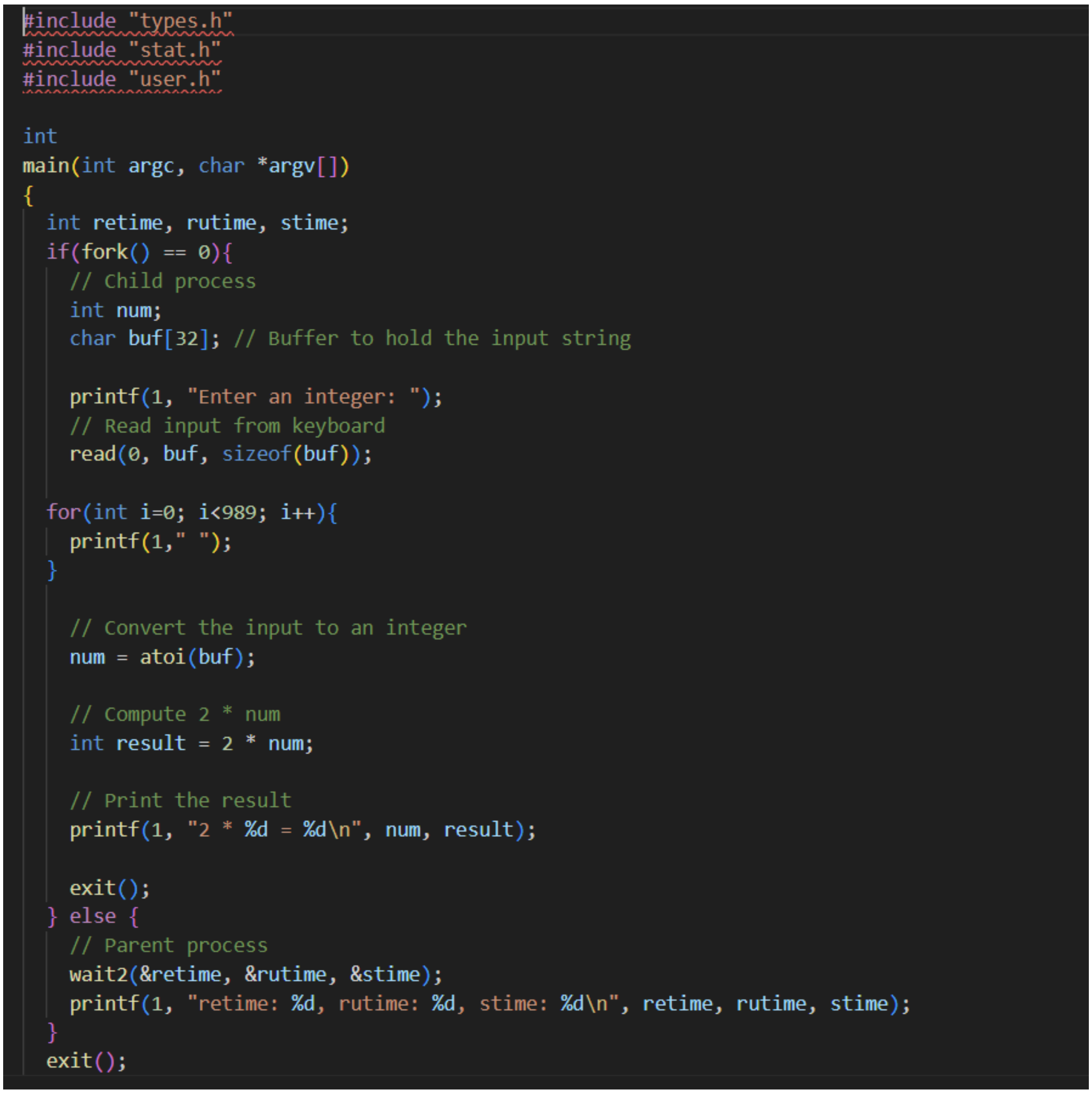
```
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(wait2)
```

We added SYSCALL(wait2) in usys.s



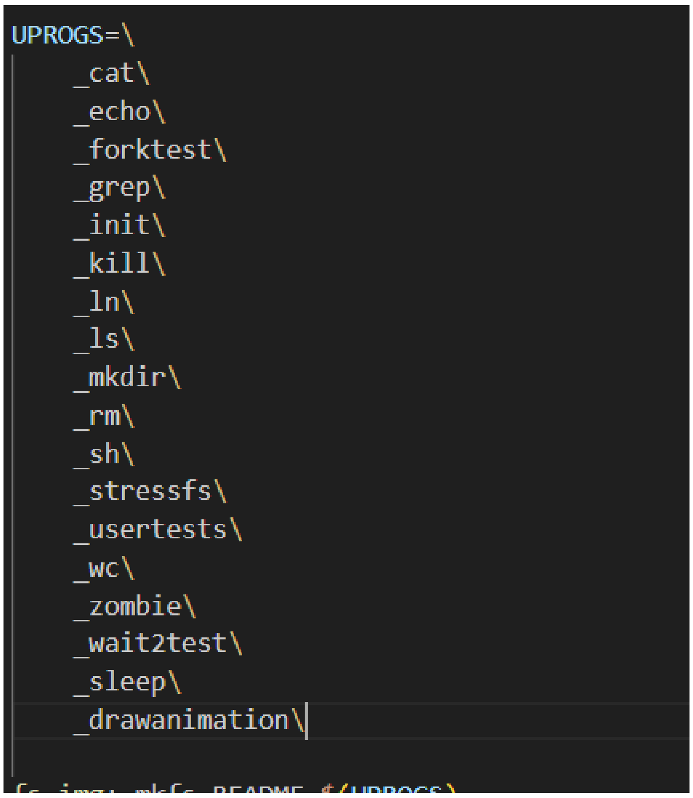
```c
char* gets(char*, int max);
uint strlen(const char*);
void* memset(void*, int, uint);
void* malloc(uint);
void free(void*);
int atoi(const char*);
int wait2(int*retime, int*rutime, int*stime);
```

We added wait2 in user.h



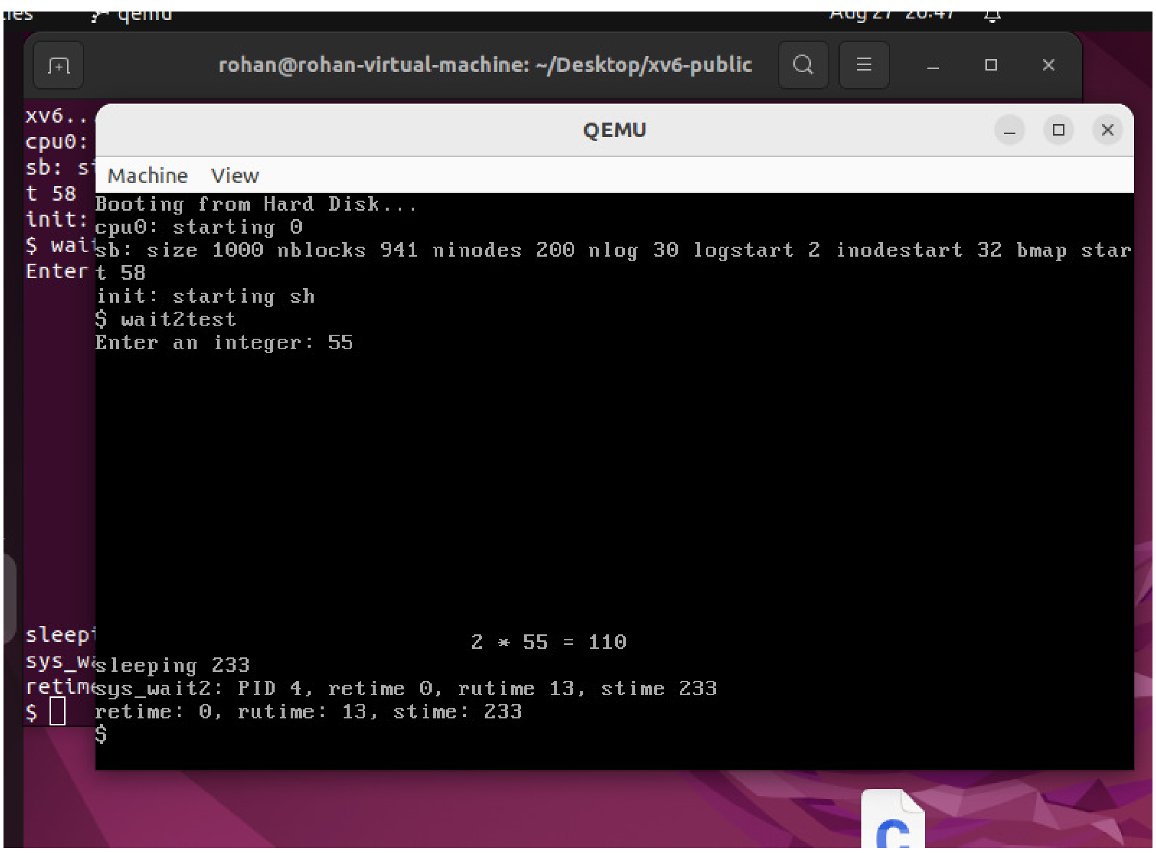
```c
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  int retime, rutime, stime;
  if(fork() == 0){
    // Child process
    int num;
    char buf[32]; // Buffer to hold the input string

    printf(1, "Enter an integer: ");
    // Read input from keyboard
    read(0, buf, sizeof(buf));

  for(int i=0; i<989; i++){
    printf(1," ");
  }

    // Convert the input to an integer
    num = atoi(buf);

    // Compute 2 * num
    int result = 2 * num;

    // Print the result
    printf(1, "2 * %d = %d\n", num, result);

    exit();
  } else {
    // Parent process
    wait2(&retime, &rutime, &stime);
    printf(1, "retime: %d, rutime: %d, stime: %d\n", retime, rutime, stime);
  }
  exit();
```

This is the testfile for testing the scheduling of the processes



```
UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\
	_wait2test\
	_sleep\
	_drawanimation\

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

Inside makefile we added wait2test program to UPROGS.



```
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
$ wait2test
Enter an integer: 55




                              2 * 55 = 110
sleeping 233
sys_wait2: PID 4, retime 0, rutime 13, stime 233
retime: 0, rutime: 13, stime: 233
$
```

This is the output for the testfile.