

گزارش بخش سوم فاز اول پروژه xv6:

ابتدا در فایل proc.h ساختار هر process را تغییر می‌دهیم و تعدادی field به آن اضافه می‌کنیم.

```
// Per-process state
struct proc {
    uint sz;                // Size of process memory (bytes)
    ...
    int stime;              // Process start time
    int rtime;              // Process run time
    int iotime;             // Process I/O time
    int etime;              // Process end time
};
```

همان طور که در کامنت‌ها ذکر شده، stime زمان شروع هر process، rtime زمان اجرای هر process، iotime همان مجموع زمان مربوط به IO، etime زمان خاتمه process است.

بعد از این نوبت به تعریف فراخوانی سیستمی (system call) ای به نام waitx می‌رسد.

برای این کار ابتدا در فایل syscall.h یک شماره جدید برای waitx تعریف می‌کنیم. از این شماره در آرایه‌ای که قرار است برای routine پاسخگویی به آن فراخوانی سیستمی انجام می‌دهیم.

```
// new system call number
#define SYS_waitx 22
```

آخرین شماره‌ای که در این فایل تعریف شده 21 می‌باشد. برای راحتی کار ما عدد بعدی (22) را برای فراخوانی سیستمی جدید در نظر می‌گیریم.

سپس در فایل syscall.c خط زیر را باید اضافه کنیم: (این خط برای تعریف routine پاسخگویی به آن فراخوانی سیستمی waitx است، یعنی هر گاه که برنامه سطح کاربر از سیستم عامل درخواست کرد، این تابع برای پاسخگویی به آن اجرا می‌شود)

```
// new system call routine
extern int sys_waitx(void);
```

این خط صرفاً تعریف یا signature تابع را صدا می‌کنیم.

سپس در آرایه‌ای که در آن اشاره‌گر به تابع‌هایی که routine های پاسخگویی به فراخوانی‌های سیستمی را قرار داده ایم، موردی برای فراخوانی سیستمی جدید را اضافه می‌کنیم. (این آرایه تمام routine های پاسخگویی به فراخوانی‌های سیستمی را در خود جای داده است)

```
static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
...
// new system call routine function pointer
[SYS_waitx]   sys_waitx,
};
```

سپس خط زیر را در فایل user.h زیر جایی که آخرین system call تعریف شده است اضافه می‌کنیم: (این قسمت signature تابع سیستمی جدید در سطح kernel است و درون sys\_waitx این تابع را صدا می‌زنیم)

```
// system calls
int fork(void);
...
// new system call function
int waitx(int*, int*);
```

سپس در فایل usys.S نیز اسم تابع سیستمی خود (waitx) را تعریف می‌کنیم.

SYSCALL(fork)

...

**SYSCALL(waitx)**

سپس در فایل sysproc.c یک system process برای پاسخگویی به interrupt مربوط به waitx تعریف می‌کنیم: (منظور همان روتین پاسخگویی به فراخوانی سیستمی است)

```
int sys_waitx(void)
{
    int *wtime, *rttime;
    if (argptr(0, (void*)&wtime, sizeof(wtime)) < 0)
        return -1;
    if (argptr(1, (void*)&rttime, sizeof(rttime)) < 0)
        return -1;
    return waitx(wtime, rttime);
}
```

در این قسمت ورودی ها را خوانده و آن ها را به متغیر های خود که می خواهیم آنها را تغییر دهیم نگاشت می کنیم (برای این کار از دستورات **argint** استفاده کرده ایم که ورودی را از آرگومان های فراخوانی سیستمی می خوانند، استفاده کرده ایم).

در فایل **defs.h** نیز در قسمت مربوط به **//proc.c** خط signature تابع را تعریف می کنیم:

```
//PAGEBREAK: 16
// proc.c
int      cpuid(void);
...
// new function for user level
int      waitx(int*, int*);
```

در فایل **proc.c** نیز تعریف تابع **waitx** را اضافه می کنیم:

```
int
waitx(int *wtime, int *rtime)
{
    struct proc *p;
    int havekids, pid;
    struct proc *curproc = myproc();

    acquire(&ptable.lock);
    for(;;){
        // Scan through table looking for exited children.
        havekids = 0;
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p->parent != curproc)
                continue;
            havekids = 1;
            if(p->state == ZOMBIE){
                // Found one.
                pid = p->pid;
                kfree(p->kstack);
                p->kstack = 0;
                freevm(p->pgdir);
                p->pid = 0;
                p->parent = 0;
                p->name[0] = 0;
                p->killed = 0;
                p->state = UNUSED;
                *wtime=p->etime-p->stime-p->rtime;
                *rtime=p->rtime;
                release(&ptable.lock);
            }
        }
    }
}
```

```

        return pid;
    }
}

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

// Wait for children to exit. (See wakeup1 call in proc_exit.)
sleep(curproc, &ptable.lock); //DOC: wait-sleep
}
}

```

در این قسمت در ابتدا همه پردازها را چک کرده و در صورتی که یک پردازه پیدا کردیم که parent آن پردازه فعلی در حال اجرا باشد، اطلاعات خواسته شده در فراخوانی سیستمی را باز می‌گردانیم.

برای آپدیت نگه داشتن زمان‌ها برای هر فرآیند، تابع زیر را می‌نویسیم (در فایل proc.c):

```

void updateProcessTimes()
{
    struct proc *p;
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
        switch(p->state) {
            case SLEEPING:
                p->iotime++;
                break;
            case RUNNABLE:
                p->iotime++;
                break;
            case RUNNING:
                p->rtime++;
                break;
            default:
                ;
        }
    }
    release(&ptable.lock);
}

```

و سپس در فایل trap.c در تابع trap، در جایی که clock یا همان ticks را آپدیت می‌کنیم، آن را صدا می‌کنیم:

```
//PAGEBREAK: 41
```

```

void
trap(struct trapframe *tf)
{
    if(tf->trapno == T_SYSCALL){
        ...
        ticks++;
        //-----
        //updating times for the process
        updateProcessTimes();

        //-----
        ...
    }
}

```

برای تست برنامه نوشته شده نیز یک فایل جدید ایجاد می‌کنیم تا بتوانیم دستور جدیدی که می‌خواهیم در کرنل تعریف کنیم را در آن قرار دهیم و آن را تست کنیم.

پس یک فایل جدید به نام **WX.C** ایجاد می‌کنیم و قطعه کد زیر را در آن قرار می‌دهیم:

```

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

int wx(int* wtime, int* rtime)
{
    return waitx(wtime,rtime);
}

int main(int argc, char *argv[])
{
    int wtime;
    int rtime;

    int a = fork();

    if(a>0){
        int npid=fork();
        if(npid==0){
            malloc( 1710 * (sizeof(int)));
            exit();
        }
    }
}

```

```

int nnpid=fork();
if(nnpid==0){
    malloc(1750 * (sizeof(int)));
    exit();
}

wx(&wtime,&rtime);
printf(1,"\n*****in user level*****\n");
printf(1,"wait time= %d, run time= %d\n ",wtime,rtime);
wx(&wtime,&rtime);
printf(1,"wait time= %d, run time= %d\n ",wtime,rtime);
wx(&wtime,&rtime);
printf(1,"wait time= %d, run time= %d\n ",wtime,rtime);
wx(&wtime,&rtime);
printf(1,"wait time= %d, run time= %d\n ",wtime,rtime);

exit();
}
else
{
    if(fork()){
        malloc(1700 * (sizeof(int)));
        if (fork())
        {
            malloc(1008 * (sizeof(int)));
            wait();
            wait();
            exit();
        }
        else
        {
            malloc(1800 * (sizeof(int)));
            exit();
        }
    }else{
        malloc(1090 * (sizeof(int)));
        if (fork())
        {
            malloc( 1030 * (sizeof(int)));
            wait();
            exit();
        }
        else
        {

```

```

        malloc(1100 * (sizeof(int)));
        exit();
    }
}
}
}

```

در قطعه کد بالا تعدادی پردازنده جدید ایجاد کرده (زیرا تعداد پردازنده‌های خود سیستم عامل xv6 زیاد نیست) و سپس فراخوانی سیستمی که آن را در تابع wx قرار داده‌ایم صدا می‌کنیم. به خاطر اینکه در سطح user هستیم از دستور printf استفاده کرده ایم که آرگومان اول آن جایی است که باید آرگومان دوم در آن نوشته شود. ما نیز 1 را که همان stdout است به تابع پاس داده-ایم و این یعنی آرگومان دوم را روی کنسول نمایش می‌دهد.

در آخر نیز برای کامپایل کردن کد و نیز لینک کردن object های ایجاد شده توسط کامپایلر، خطوط زیر را باید در قسمت های گفته شده به MakeFile موجود در پروژه اضافه کنیم:

```

UPROGS=\
    _cat\
    ...
    _wx\

```

```

EXTRA=\
    mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
    ...
    wx.c\

```

سپس برای امتحان کردن برنامه نوشته شده باید دستورات زیر را اجرا کنیم:

make clean

make qemu

بعد از اجرای این دستورات shell مربوط به سیستم عامل xv6 باز می‌شود و ما می‌توانیم دستور جدید و برنامه خود را تست کنیم.

وقتی که shell باز شد باید دستور زیر را در آن وارد کنیم:

wx

نمونه‌هایی از خروجی‌های برنامه در زیر آورده شده است:

```
$ wx  
  
*****in user level*****  
wait time= 1, run time= 0  
wait time= 2, run time= 2  
wait time= 2, run time= 0  
wait time= 2, run time= 0
```

```
wait time= 2, run time= 0  
$ wx  
  
*****in user level*****  
wait time= 1, run time= 0  
wait time= 1, run time= 2  
wait time= 1, run time= 0  
wait time= 1, run time= 0  
$ _
```