操作系统实验报告

实验七 作业调度

学号	姓名
10231016	童浩（组长）
10231008	解佳琦
10231024	陈宇宁
38230213	邱伟豪

1 需求说明

1.1 基本需求

本实验要求实现一个作业调度程序，通过该程序可以完成作业的入队、出队、查看和调度。具体要求如下：

（1）实现作业调度程序scheduler，负责整个系统的运行。

这是一个无限循环运行的进程，其任务是响应作业的入队、出队以及状态查看请求，采用适当的算法调度各作业运行。

（2）实现作业入队命令。

1.1.1 基本命令的格式及参数说明

本程序设计的基本格式及参数说明如下：

1. 实现作业入队命令

   格式：enq [-p num] e_file [args]

   参数说明：

   -p num：设定作业的初始优先级，范围为0～3且默认值为0；

   e_file：启动作业执行的可执行文件（以/开始的绝对路径名）。

   args：e_file 的运行参数。

   用户通过该命令给scheduler 发送入队请求，将作业提交给系统运行。每一个作业提交后，若创建成功，scheduler 都将为其分配一个唯一标识jid。Scheduler 调度程序为每个作业创建一个进程，并将其状态置为READY，然后放入就绪队列中，打印作业信息。

2. 实现作业出队命令

   格式：deq jid

   参数说明：

jid：由 scheduler 分配的作业号。

用户通过该命令给scheduler 发送出队请求，scheduler 将使该作业出队，然后清除相关的数据结构。若该作业正在运行，则需先终止其运行。每个用户都只能杀掉（kill）自己提交的作业。

2. 实现作业状态查看命令

格式：stat

参数说明：

在标准输出上打印出就绪队列中各作业的信息。状态信息应该包括：

- 作业的jid；
- 作业提交者用户名；
- 作业执行的时间；
- 在就绪队列中的等待时间；
- 作业创建的时刻；
- 此时作业的状态（READY、RUNNING）。

3. 实现多级反馈的轮转调度算法

每个作业有其动态的优先级，在用完分配的时间片后，可以被优先级更高的作业抢占运行。就绪队列中的进程等待时间越长，其优先级越高。每个作业都具有以下两种优先级：

- 初始优先级（initial priority）：在作业提交时指定，将保持不变，直至作业结束。
- 当前优先级（current priority）：由scheduler 调度更新，用以调度作业运行。scheduler总是选择当前优先级最高的那个作业来运行。

作业当前优先级的更新主要取决于以下两种情况：

- 一个作业在就绪队列中等待了若干个时间片（如5 个），则将它的当前优先级加1（最高为3）。
- 若当前运行的作业时间片到，则中止该作业停运行（抢占式多任务），将其放入就绪队列中，它的当前优先级也恢复为初始优先级。

通过这样的反馈处理，使得每个作业都有执行的机会，避免了使低优先级的作业拖延而不能执行的情况发生。

出于简单的目的，假设只考虑作业的两种状态：

- READY：就绪状态，该作业在就绪队列（ready queue）中等待调度。
- RUNNING：运行状态，该作业正在运行。

1.2 进阶需求

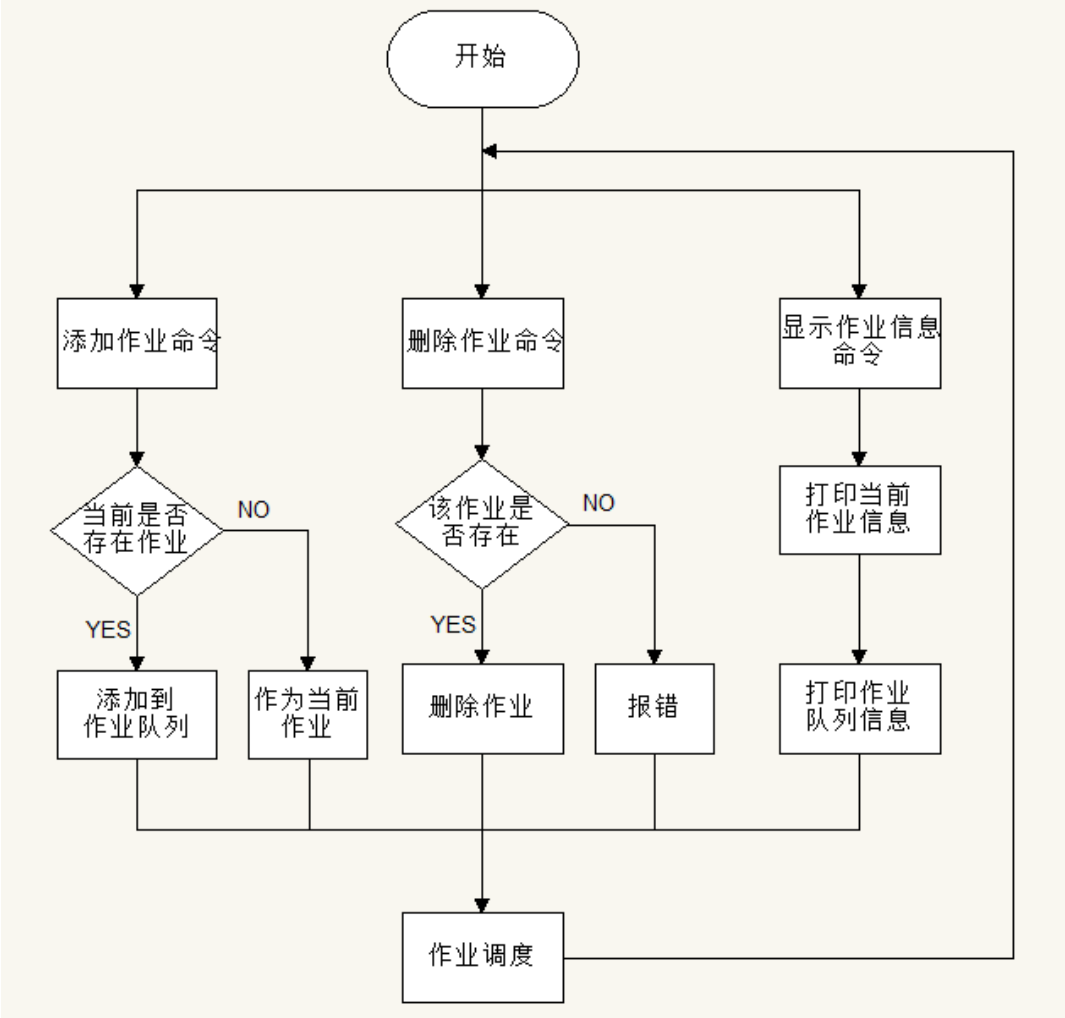
1. 实现多级轮转调度，优化优先级顺序和时间片分配。

2. 改进stat命令显示方式，利用FIFO将作业状态信息显示在新的窗口下。

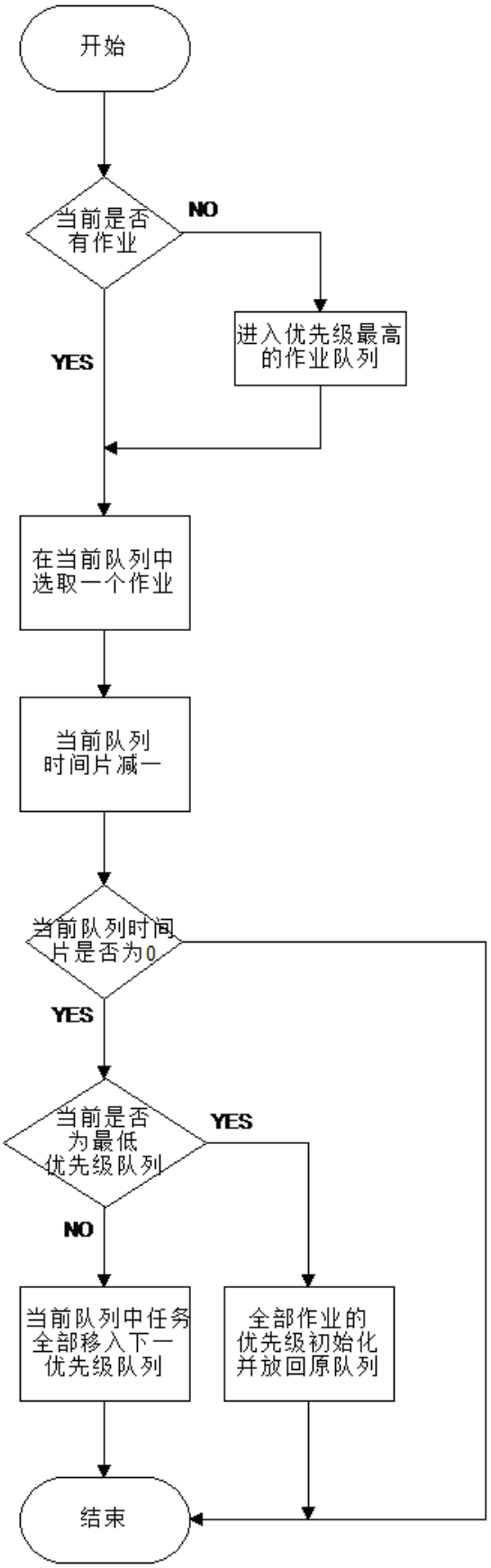
2 设计说明

2.1 结构设计

2.1.1 程序流程图



2.1.2 作业调度流程


开始
当前是否有作业
NO
YES
进入优先级最高的作业队列
在当前队列中选取一个作业
当前队列时间片减一
当前队列时间片是否为0
YES
当前是否为最低优先级队列
YES
NO
当前队列中任务全部移入下一优先级队列
全部作业的优先级初始化并放回原队列
结束

2.2 功能设计

这里给出各功能模块的设计方案或实现方法。

2.2.1 重要的数据结构设计

```
struct jobcmd{
        enum cmdtype type;
        int argnum;
        int owner;
        int defpri;
        char data[BUFLEN];
};

struct jobinfo{
        int jid;                    /*  作业 ID */
        int pid;                    /*  进程 ID */
        char** cmdarg;              /*  命令参数  */
        int defpri;                 /*  默认优先级  */
        int curpri;                 /*  当前优先级  */
        int ownerid;                /*  作业所有者 ID */
        int wait_time;              /*  作业在等待队列中等待时间  */
        time_t create_time;         /*  作业创建时间  */
        int run_time;               /*  作业运行时间  */
        enum jobstate state;        /*  作业状态  */
};

struct waitqueue{
        struct waitqueue *next;
        struct jobinfo *job;
};
```

2.2.2 主要函数或接口设计

这里给出主要函数或接口的功能说明、实现方法和调用关系。

2.2.2.1 函数功能说明

1. 文件 deq.c 中

函数名称：int main(int argc,char *argv[])

函数功能：添加新作业的主函数
参数说明：main 函数的基本参数

函数名称：void usage()
函数功能：打印语句用法
参数说明：无

2. 文件 enq.c 中
函数名称：int main(int argc,char *argv[])
函数功能：终止作业的主函数
参数说明：main 函数的基本参数

函数名称：void usage()
函数功能：打印语句用法
参数说明：无

3. 文件 error.c 中
函数名称：void error_doit(int errnoflag,const char *fmt,va_list ap)
函数功能：
参数说明：

函数名称：void error_sys(const char *fmt,...)
函数功能：
参数说明：

函数名称：void error_quit(const char *fmt,...)
函数功能：
参数说明：

函数名称：void error_msg(const char *fmt,...)
函数功能：
参数说明：

4. 文件 job.c 中
函数名称：void scheduler()
函数功能：初始化作业调度程序
参数说明：无

函数名称：int allocjid()
函数功能：返回下一个工作 id
参数说明：无

函数名称：void updateall()
函数功能：更新所有作业的运行时间等待时间及优先级等

参数说明：无

函数名称：struct waitqueue* jobselect()
函数功能：遍历等待队列中的作业，找到优先级最高的作业
参数说明：无

函数名称：void jobswitch()
函数功能：切换作业队列
参数说明：无

函数名称：void sig_handler(int sig,siginfo_t *info,void *notused)
函数功能：处理作业之间的信号
参数说明：sig 为需要处理的信号类型，info 为信号内容信息，notused 为无用信息

函数名称：void do_enq(struct jobinfo *newjob,struct jobcmd enqcmd)
函数功能：向等待队列中增加新的作业
参数说明：jobinfo *newjob 为新添加作业的信息，jobcmd enqcmd 为添加作业的命令

函数名称：void do_deq(struct jobcmd deqcmd)
函数功能：在队列中查找并终止作业
参数说明：jobcmd deqcmd 为终止作业的命令

函数名称：void do_stat(struct jobcmd statcmd)
函数功能：显示作业的统计信息
参数说明：jobcmd statcmd 为显示统计信息的命令

函数名称：int main()
函数功能：作业调度的主函数
参数说明：无

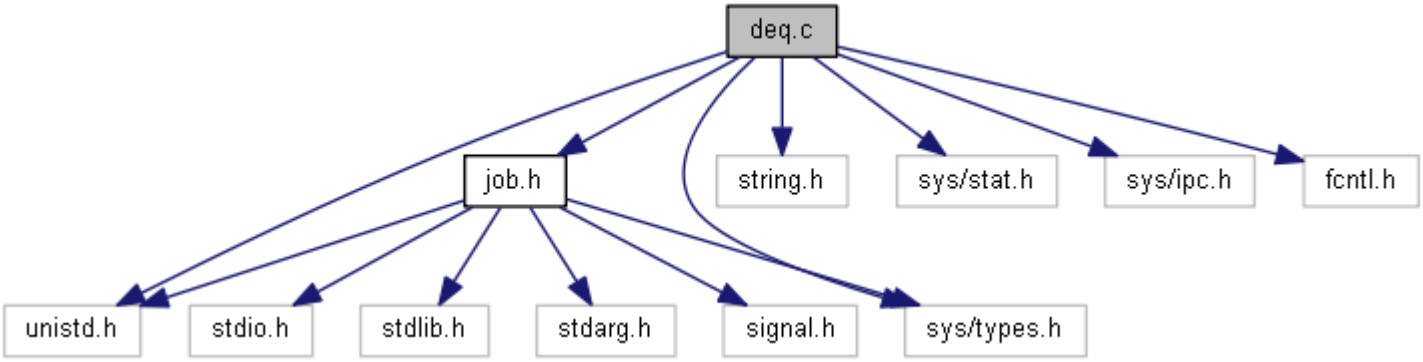
5. 文件 stat.c 中
函数名称：int main(int argc,char *argv[])
函数功能：显示统计信息功能的主函数
参数说明：main 函数的基本参数

函数名称：void usage()
函数功能：打印语句用法
参数说明：无

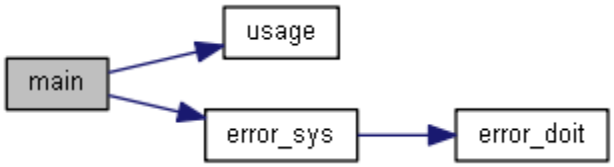
2.2.2.2 文件引用及函数调用关系

1. deq.c

文件引用关系

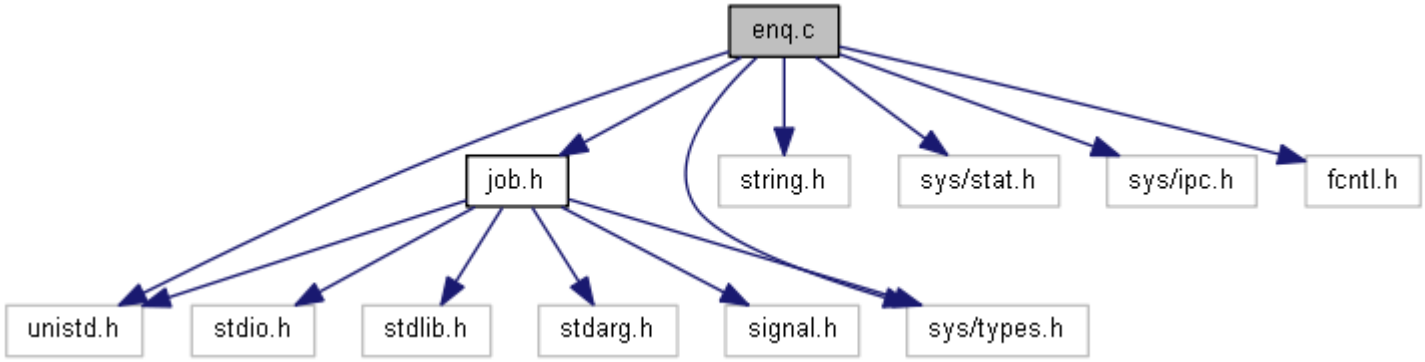


函数调用关系

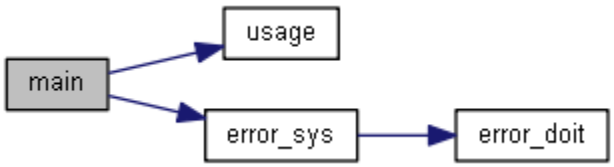


2. enq.c

文件引用关系

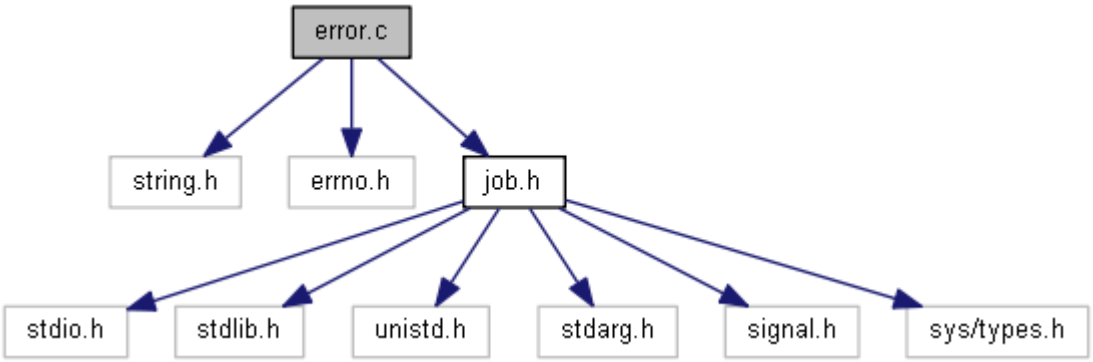


函数调用关系

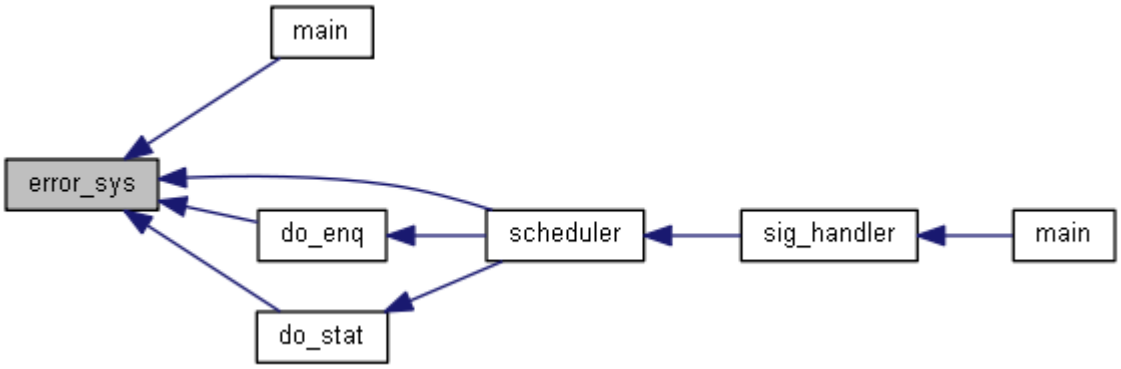


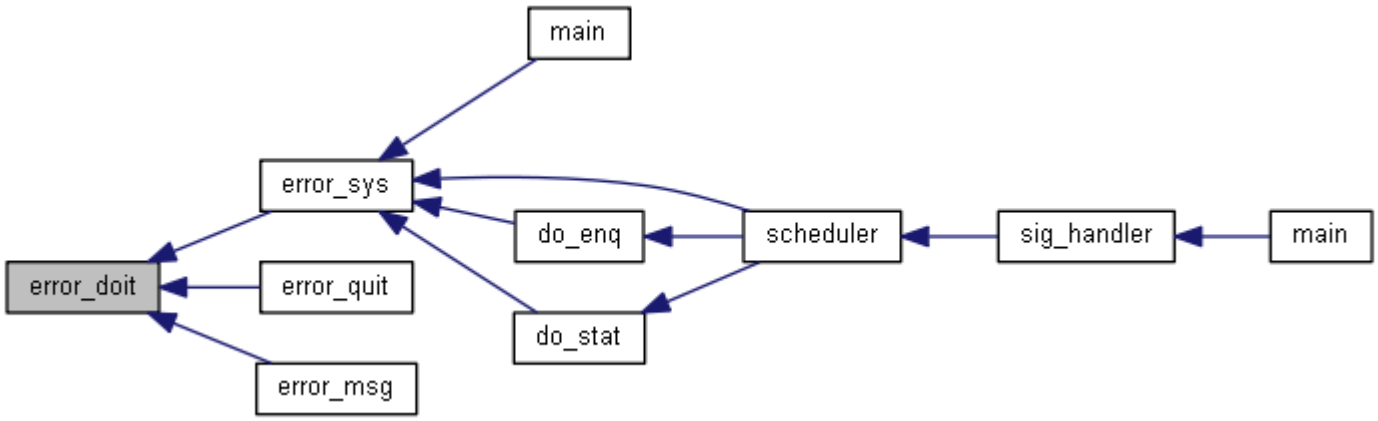
3. error.c

文件引用关系



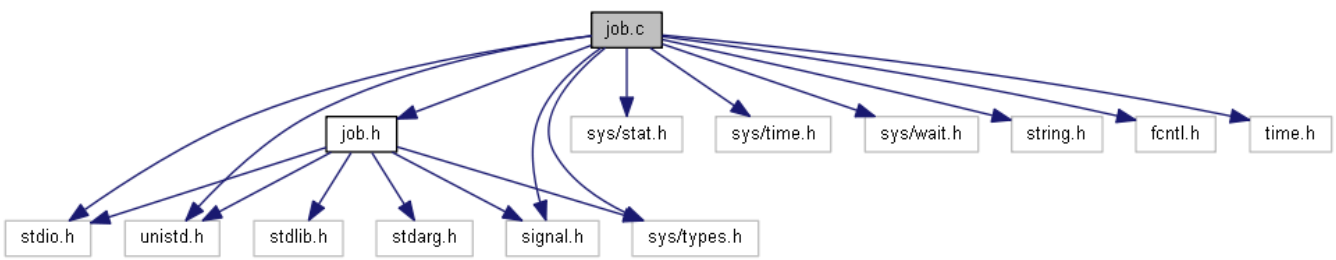
函数调用关系



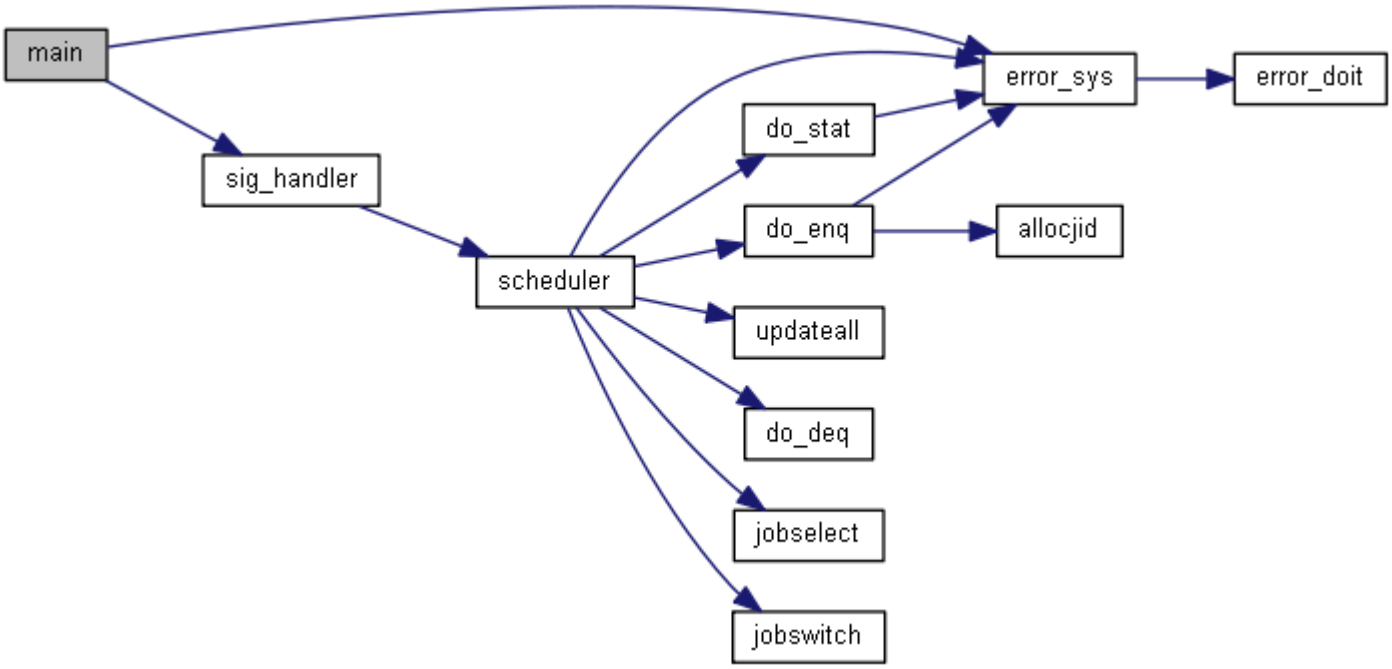


4. job.c

文件引用关系

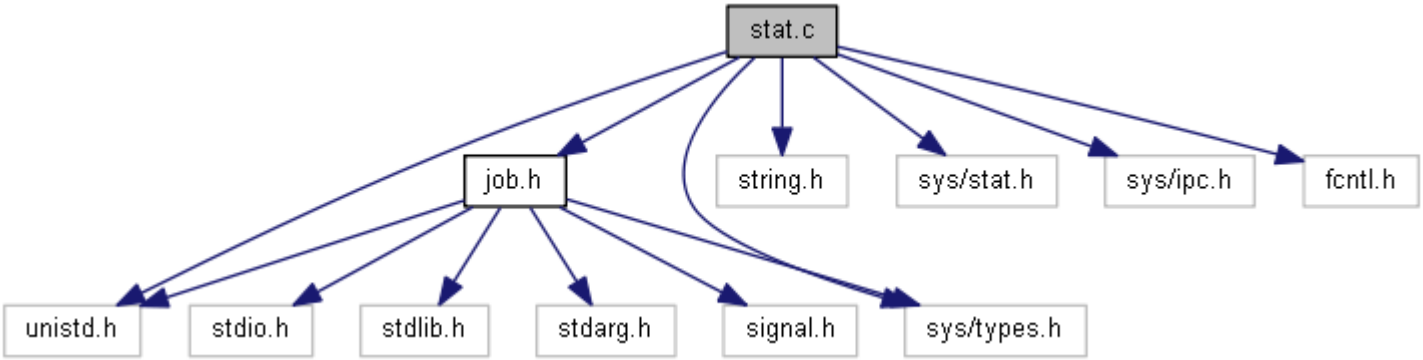


函数调用关系

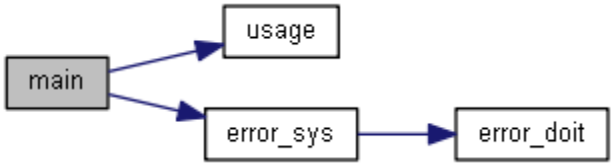


5. stat.c

文件引用关系



函数调用关系



2.2.3 主要算法说明

2.2.3.1 作业调度算法

```
void jobswitch()
{
	struct waitqueue *p;
	int i;

	if(current && current->job->state == DONE){ /* 当前作业完成 */
		/* 作业完成，删除它 */
		for(i = 0;(current->job->cmdarg)[i] != NULL; i++){
			free((current->job->cmdarg)[i]);
			(current->job->cmdarg)[i] = NULL;
		}
		/* 释放空间 */
		free(current->job->cmdarg);
		free(current->job);
		free(current);

		current = NULL;
	}

	if(next == NULL && current == NULL) /* 没有作业要运行 */
		return;
	else if (next != NULL && current == NULL){ /* 开始新的作业 */

		printf("begin start new job\n");
		current = next;
		next = NULL;
		current->job->state = RUNNING;
		kill(current->job->pid,SIGCONT);
		return;
	}
```

```
    else if (next != NULL && current != NULL){ /*  切换作业  */

        printf("switch to Pid: %d %d %d\n",next->job->pid,currentpri,next->job->curpri);
        kill(current->job->pid,SIGSTOP);
        //current->job->curpri = current->job->defpri;
        current->job->wait_time = 0;
        current->job->state = READY;

        current->next = NULL;

        /*  放回等待队列  */
        if(head != NULL){
            for(p = head; p->next != NULL; p = p->next);
            p->next = current;
        }else{
            head = current;
        }
        current = next;
        next = NULL;
        current->job->state = RUNNING;
        current->job->wait_time = 0;
        kill(current->job->pid,SIGCONT);
        return;
    }else{ /* next == NULL 且 current != NULL，不切换  */
        return;
    }
}
```

1、进程在进入待调度的队列等待时，首先进入优先级最高的 Q1 等待。
2、首先调度优先级高的队列中的进程。
3、对于同一个队列中的各个进程，按照时间片轮转法调度。
4、在低优先级的队列中的进程在运行时，又有新到达的作业，那么在运行完这个时间片后，CPU 马上分配给新到达的作业（抢占式）。

2.2.3.1 stat 打印信息算法

```
//job.c 中
```

```
sprintf(fifoname,"/tmp/stat-tmp-uid-%d",statcmd.owner);
    if(stat(fifoname,&statbuf)==0){
        /* 如果 FIFO 文件存在,删掉 */
        if(remove(fifoname)<0)
            error_sys("remove failed");
    }

    if(mkfifo(fifoname,0666)<0)
        error_sys("mkfifo failed");
//建立新的 FIFO，用于将信息发送给 stat 进程
//中间省略

if(write(fd,"finished",BUFLEN*3)<0)
        error_sys("stat write failed");
    close(fd);

//所有信息发送完后，发送 finished 字段，表示结束
```

```
//stat.c 中

    sprintf(fifoname,"/tmp/stat-tmp-uid-%d",getuid());
    while(1){
        if(stat(fifoname,&statbuf)==0)
            break;
    }
    //从 statfifo 读入
    if((statfifo=open(fifoname,O_RDONLY|O_NONBLOCK))<0)
        error_sys("open fifo failed");

    while(!(read(statfifo,statstr,BUFLEN*3)>0)){
        //printf("%s",statstr);
    }
    printf("%s",statstr);
//首先为对应的用户打开对应的 FIFO 文件，用于读取。
while(1){
        read(statfifo,statstr,BUFLEN*3);
        if(strcmp(statstr,"finished")==0)
            break;
        printf("%s",statstr);
}
//循环读取并打印直到读到 finished
```

```
close(statfifo);
    if(stat(fifoname,&statbuf)==0){
        /* 如果 FIFO 文件存在,删掉 */
        if(remove(fifoname)<0)
            error_sys("remove failed");
    }
//最后关闭 FIFO 并删除临时文件
```

3 测试和使用说明

3.1 使用说明

列出程序的开发环境，如操作系统、使用的编程语言、开发工具和组件等。

列出程序的运行环境，如操作系统、必要的运行库等。

开发环境：

操作系统：Ubuntu12.04

编程语言：C 语言

开发工具：CodeBlocks

运行环境：

操作系统：Ubuntu 12.04

运行库：Bison、Flex、Gcc

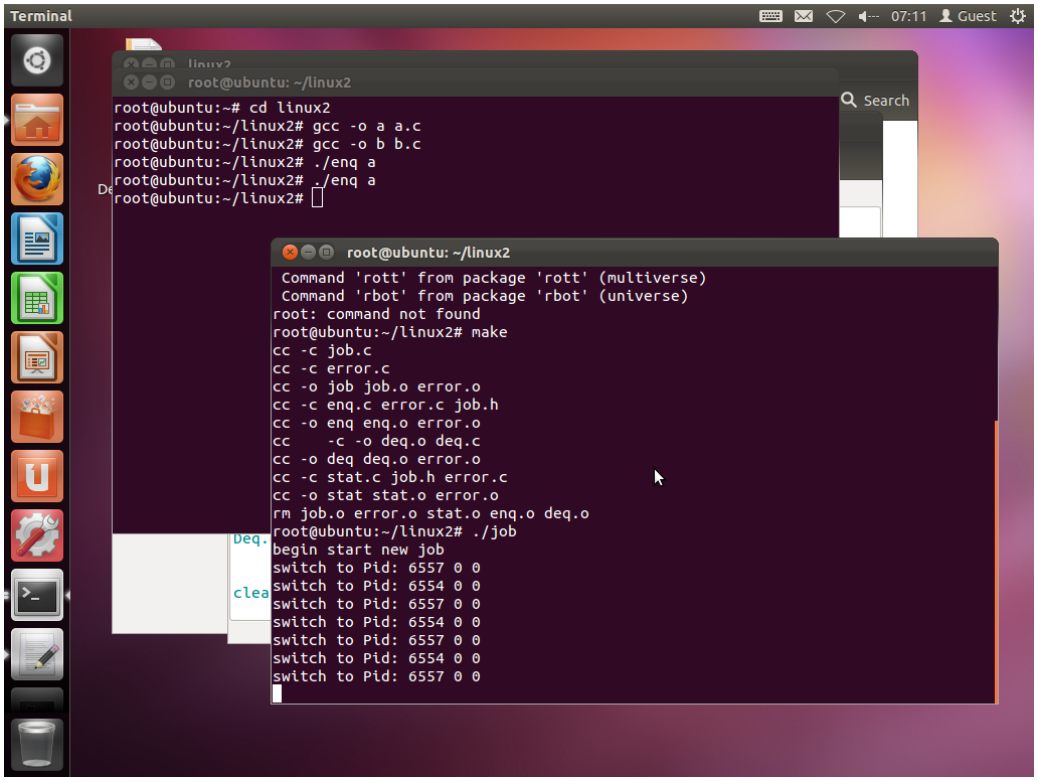
3.2 测试说明

3.2.1 添加任务

命令： ./enq a

./enq a

运行结果：

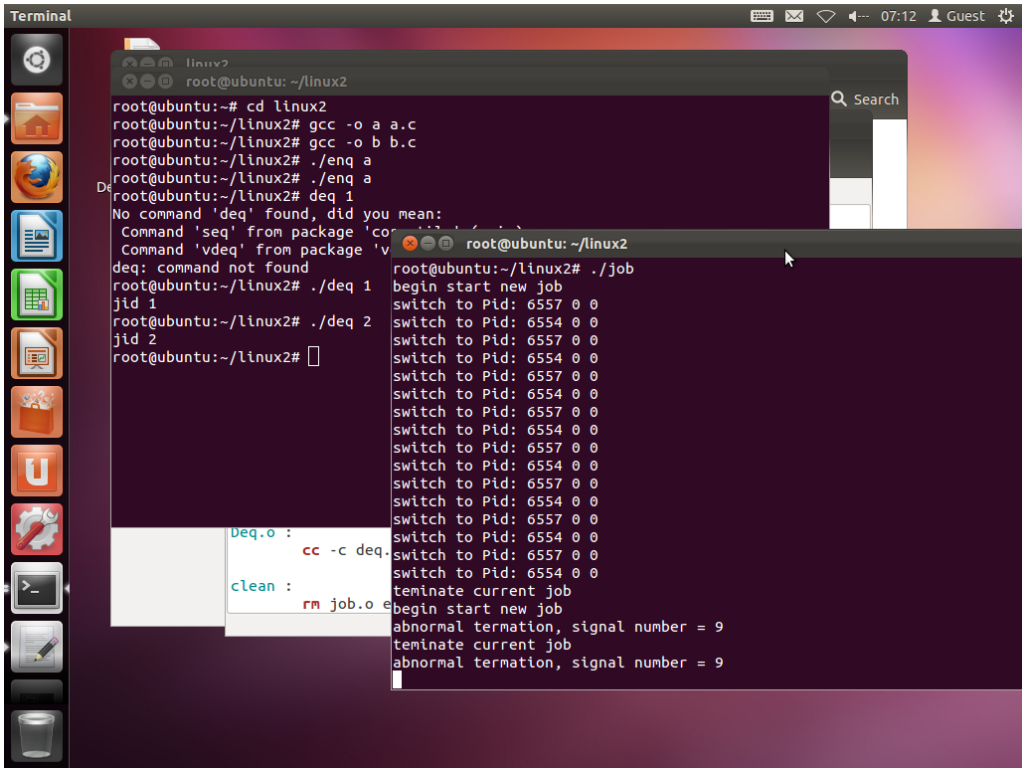


3.2.2 删除任务

命令： ./deq 1

./deq 2

运行结果：



3.2.3 调度算法的测试

命令： ./enq a

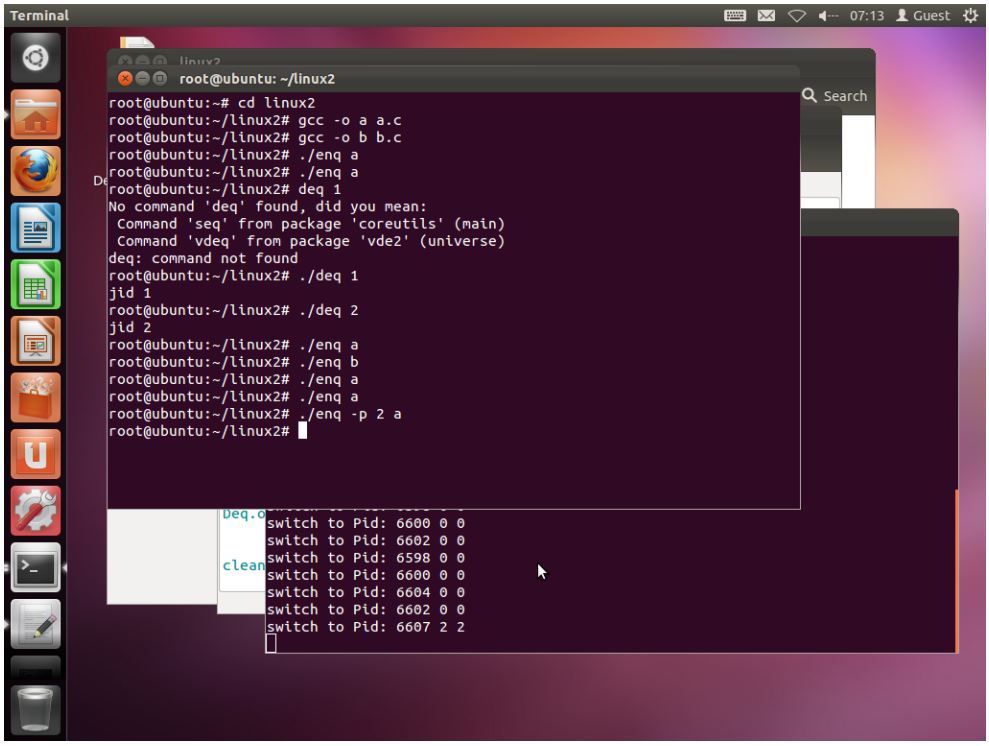
./enq b

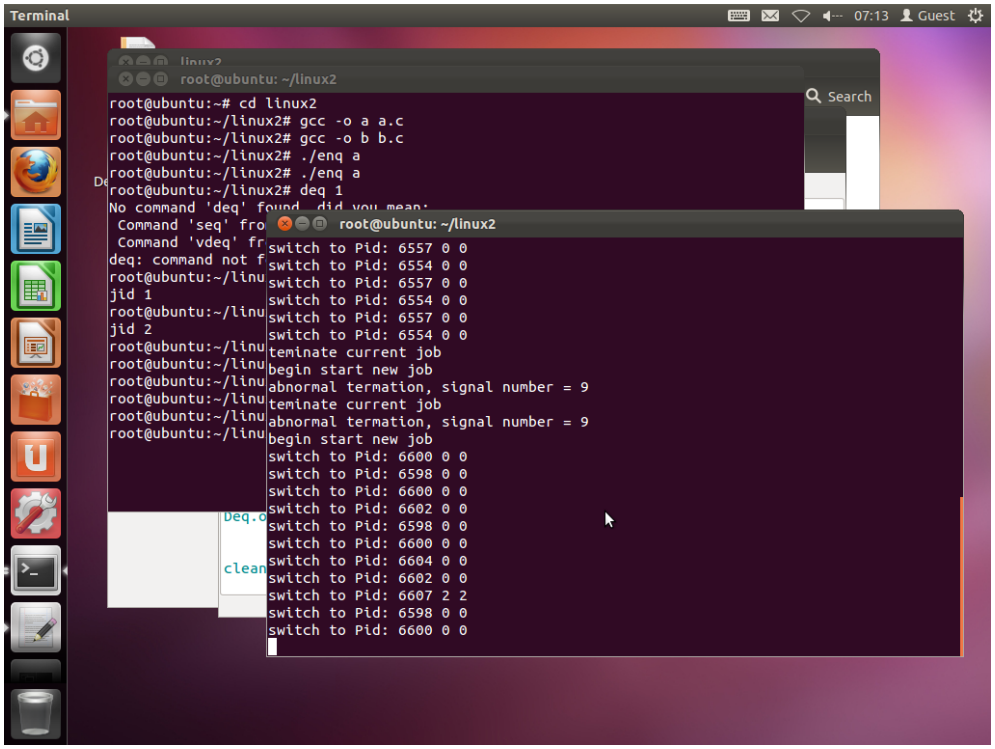
./enq a

./enq a

./enq -p 2 a

运行结果：

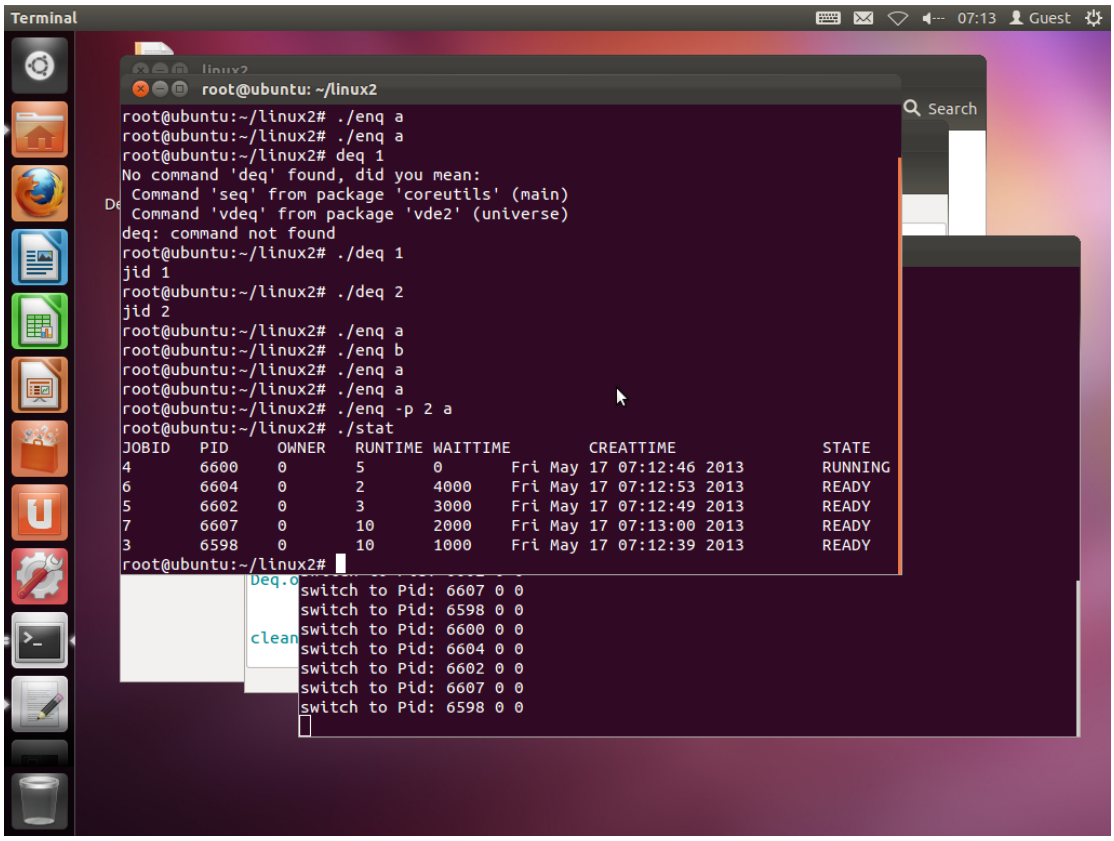




3.4.4 显示任务信息

命令： ./stat

运行结果：



4 会议记录

（1） 第一次会议

组员	前一阶段任务	是否完成	后一阶段任务
童浩	确定组员分工，学习 Linux 作业调度相关的知识，进行阅读并熟悉源代码等工作	是	多级反馈轮转算法的改进
解佳琦		是	实现作业状态信息的反馈
陈宇宁		是	对源代码进行 bug 修正
邱伟豪		是	对源代码进行 bug 修正

（2） 第二次会议

组员	前一阶段任务	是否完成	后一阶段任务
童浩	多级反馈轮转算法的改进	是	整合源代码并调试
解佳琦	实现作业状态信息的反馈	是	整合源代码并调试
陈宇宁	对源代码进行 bug 修正	是	完成相关文档内容
邱伟豪	对源代码进行 bug 修正	是	完成相关文档内容

5 其它说明

学号	姓名	分工情况	工作量比例
10231016	童浩（组长）	改进多级反馈轮转算法的更新方式	30%
10231008	解佳琦	实现作业状态信息的反馈	26%
10231024	陈宇宁	对源代码进行 bug 修正	24%
38230213	邱伟豪	对源代码进行 bug 修正	20%

6 程序清单

6.1 源代码

（路径：~\源代码\linux2）

6.1.1 程序源文件

compile.sh
deq.c
enq.c
error.c
job.c
job.h
makefile
stat.c

6.2.1 测试用程序

a.c　b.c

6.2 可执行程序

（路径：~\可执行程序\）
deq
enq
job

stat

6.2.1 测试用程序

a：始终循环。
b：倒计时 5 然后结束程序。

6.2 其他目录

无。