



به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر
درس آزمایشگاه سیستم عامل

پروژه چهارم

نام و نام خانوادگی	دانیال سعیدی(810198571) سروش صادقیان(810898048) محمد قره حسنلو(810198461)
تاریخ ارسال گزارش	
رپیو گیتهاب	
آخرین Commit ID	

فهرست گزارش سوالات

سوال 1.....2

سوال 2.....3

سوال 3.....3

سوال 4.....3

سوال 5.....3

سوال ۱

اگر یک lock را acquire کنیم این lock متعلق به پراسس فعلی است و اگر سایر پراسیس ها بخواهند آن lock را دریافت کنند نمیتوانند. حال فرض کنید در حین اجرای پراسس یک interrupt رخ میدهد که این منجر شود interrupt handler مربوط آن سیگنال وسط ناحیه بحرانی ما فعال شود و بخواهد به lock دسترسی پیدا کند (قابلیت انجام دادن این کار را دارد) این اتفاق خوبی نیست بنابراین interrupt handlers موقتاً غیر فعال می شود و بعد از unlock مجدداً فعال میشود. pushcli وقفه ها را غیرفعال میکند و به ازای هر cpu متغیر ncli را یک واحد افزایش میدهد. Popcli شمارنده را کاهش میدهد و اگر شمارنده صفر شود وقفه ها را فعال میکند. این توابع به این دلیل نوشته شده اند که وقفه ها فقط زمانی غیر فعال می شوند که همه قفل ها آزاد شده باشند. دو تابع (CLI)(Clear Interrupt و (STI)(Set Interrupt با فلگ IF کار دارند. این توابع فقط single processor جوابگو هستند. فلگ IF مشخص میکند که آیا CPU سریع به maskable hardware interrupts پاسخ دهد یا خیر.

```
void
popcli(void)
{
    if(readeflags() & FL_IF)
        panic("popcli - interruptible");
    if(--mycpu()->ncli < 0)
        panic("popcli");
    if(mycpu()->ncli == 0 && mycpu()->intena)
        sti();
}
```

```
void
pushcli(void)
{
    int eflags;

    eflags = readeflags();
    cli();
    if(mycpu()->ncli == 0)
        mycpu()->intena = eflags & FL_IF;
    mycpu()->ncli += 1;
}
```

سوال ۲

در روش spinlock پراسسی که منتظر است که lock آزاد شود حلقه را تکرار میکند و باعث می شود که زمان cpu صرف شود. sleeplock باعث می شود که پراسس sleep شود و تا هنگامی که نوبت آن نرسیده از زمان cpu استفاده نمیکند. هنگامی که یک دارنده sleeplock آن را رها میکند پراسسی که spi-lock of sleeplock را اول به دست آورده wakeup میکند. همچنین بقیه پراسس ها همان مکانیزم sleeplock را به کار میبرند که ترتیب waiterها را تضمین میکند. در spin-

lock شرط bounded waiting برقرار نیست. فرض کنید این حالت پیش بیاید که هنوز بازه زمانی پراسس قبل تمام نشده و دوباره همان قبلی بیاید و lock را آشغال بکند و مجدداً احتمال دارد این اتفاق تکرار شود. به این دلیل که هیچ کرانی برای اینکه پراسس رقیب چند بار وارد این حلقه می‌شود وجود ندارد پس راه حل خوبی برای مسأله producer/consumer نیست. ممکن است producer در حلقه while بماند و اجازه ندهد consumer وارد critical section شود و این مشکل اساسی است.

سوال ۳

حالت‌های مختلف یک پردازنده در xv6 عبارتند از:

Runnable, Unused, Embryo, Sleeping, Running, Zombie

وظیفه تابع sched :

هنگامی که یک پردازنده بخواهد پردازنده را آزاد کند تابع sched را فراخوانی میکند. این تابع عملیات context switch را فعال میکند. وقتی که در زمان دیگری مجدداً تعویض متن به پردازنده ی فراخواننده sched انجام شود تابع ادامه ی اجرای خود را از تابع sched فرا میگیرد. پردازنده های در موقعیت زیر پردازنده را با فراخوانی sched آزاد میکنند:

(۱) هنگامی که timer interrupt رخ میدهد که یعنی تایم کوانتم پردازنده به پایان رسیده است و نوبت را باید به پردازنده دیگری بدهیم.

(۲) زمانی که یک پردازنده کار خود را با فراخوانی تابع exit به پایان میرساند که موجب می‌شود تابع sched فراخوانی شود.

(۳) وقتی که باید به علت رخداد event به حالت sleeping برود sched را فراخوانی میکند و پردازنده را آزاد میکند.

هنگامی که یک پردازنده میخواهد پردازنده را آزاد کند باید ptable.lock را acquire کند و هر lock دیگری که دارد را آزاد کند سپس وضعیت خود را از running خارج کند

و سپس تابع sched را فراخوانی کند که این فرایند توسط sleep, yield, exit انجام میشود. Sched این شرایط را چک میکند و اگر مشکلی باشد ارور میدهد. سپس تابع switch فراخوانی میشود

سوال ۴

ساختار sleeplock در خود مقدار عددی به نام pid دارد که شناسه پردازنده نگه دارنده را در خود ذخیره میکند هنگام صدا زدن تابع aquiresleep این شناسه در lk->pid ذخیره میشود. در شکل پایین در تابع releasesleep با افزودن شرط نشان داده شده موجب میشود تنها در صورتی که پردازنده فعلی که در پردازنده در حال اجرا است و releasesleep را فراخوانی کرده است اگر شناسه یکتایی با نگهدارنده sleeplock داشته باشد عملیات آزادسازی قفل انجام شود. قفل های mutex در لینوکس کارکرد مشابهی دارند به شکلی که تسکی که میخواند وارد critical section شود باید ابتدا mutex_lock و هنگام خروج تابع mutex_unlock را فراخوانی کند. در صورتی که mutex lock قابل دسترسی نبود و توسط تسک دیگری گرفته شده بود. تسک فعلی به وضعیت sleep وارد میشود mutex lock توسط نگهدارنده آن آزاد شود. لینوکس قفل های semaphore را که بر اساس توضیحات قبل با وارد کردن تسک ها به وضعیت sleep میکنند را نیز دارد.

```
void
releasesleep(struct sleeplock *lk)
{
    acquire(&lk->lk);
    if(myproc()->pid == lk->pid) {
        lk->locked = 0;
        lk->pid = 0;
        wakeup(lk);
    }
}
```

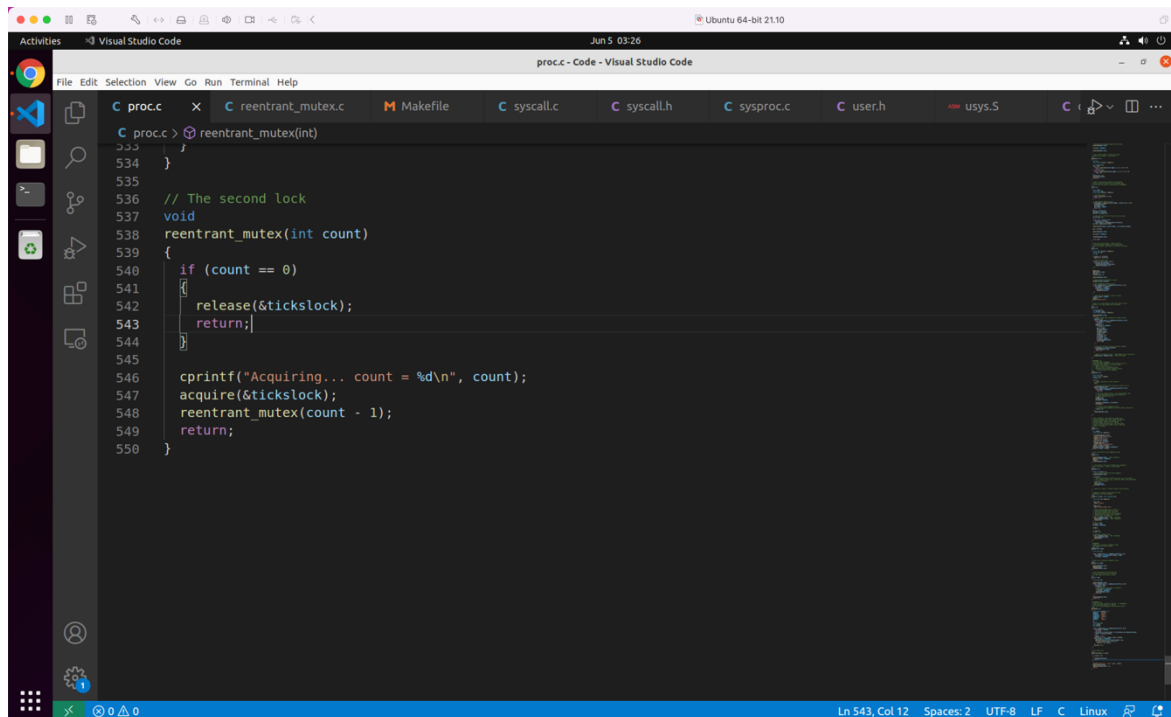
سوال ۵

حافظه تراکنشی مدلی جایگزین برای lock ها برای کنترل دسترسی به حافظه به صورت همروند در برنامه نویسی موازی میباشد. در مدل حافظه تراکنشی به جای

مشخص کردن بخش‌هایی از کد به صورت critical بخش‌هایی از کد به صورت transaction مشخص می‌شوند. هر transaction مجموعه‌ای از عملیات هاست که می‌توانند اجرا شوند و در متغیرها تا زمانی که کانفلیکت رخ نداده است تغییر ایجاد کنند. حافظه تراکنشی تضمین میکند که اجرای موازی یک برنامه چند ریسه ای معادل اجرای آن درحالتی خواهد بود که هیچکدام از بخش‌های transaction پشت سر هم انجام شوند و نه همزمان. این کار به این صورت انجام می‌شود که برخلاف lock ها حافظه تراکنشی به بخش‌های transaction اجازه میدهد که همزمان اجرا شوند اما متغیرهای مربوط به این بخش‌ها را رصد میکند. در صورتی که دو بخش تراکنش همزمان تصمیم به دسترسی به این متغیرها داشته باشند یکی از آنها abort می‌شود و به ابتدای تراکنش بازمی‌گردد که در حال انجام آن بوده و به صورت خودکار آن را از سر می‌گیرد و به این صورت دیگر برای کنترل ورود و خروج ریسه‌ها از critical section احتیاجی به lock نداریم.

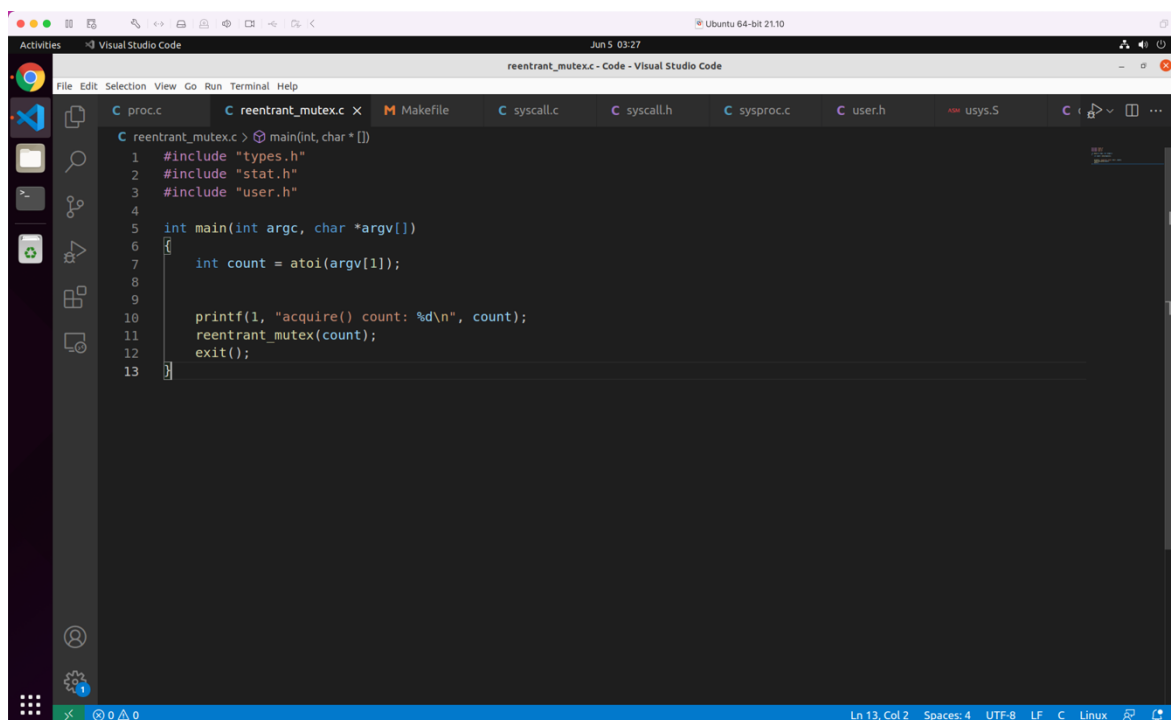
قفل با ورود مجدد

تغییرات فایل proc.c:



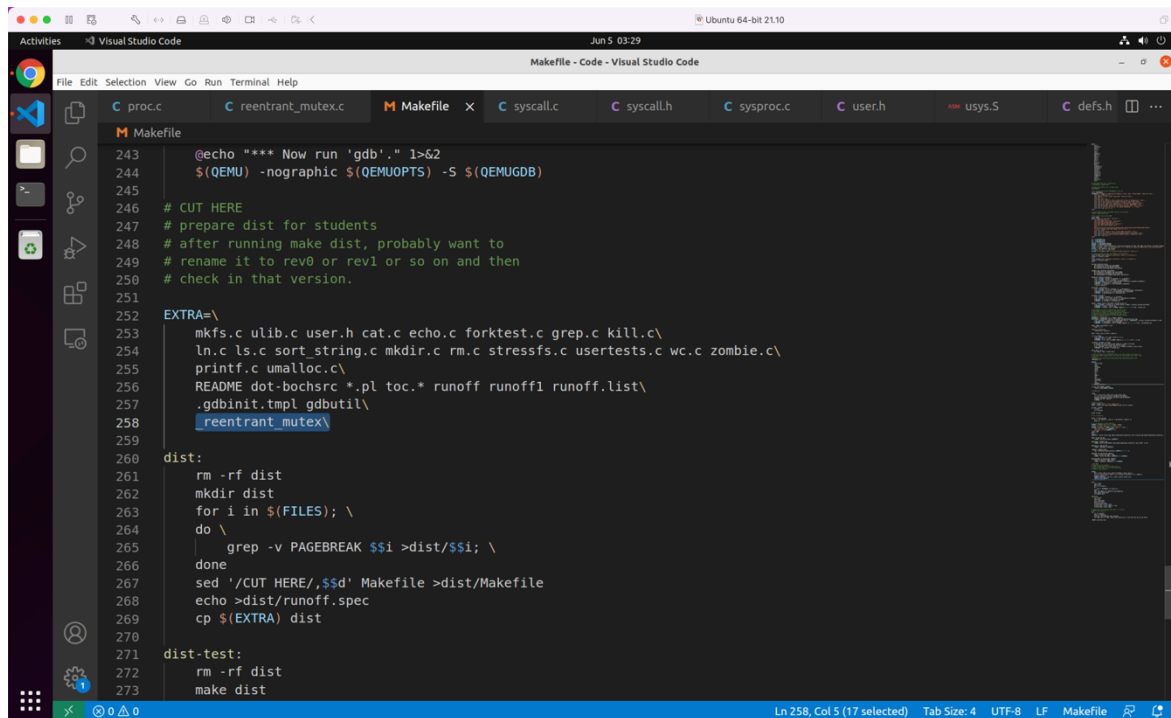
```
1  proc.c > reentrant_mutex(int)
2  }
3
4  // The second lock
5  void
6  reentrant_mutex(int count)
7  {
8      if (count == 0)
9      {
10         release(&tickslock);
11         return;
12     }
13
14     printf("Acquiring... count = %d\n", count);
15     acquire(&tickslock);
16     reentrant_mutex(count - 1);
17     return;
18 }
```

فایل reentrant_mutex.c



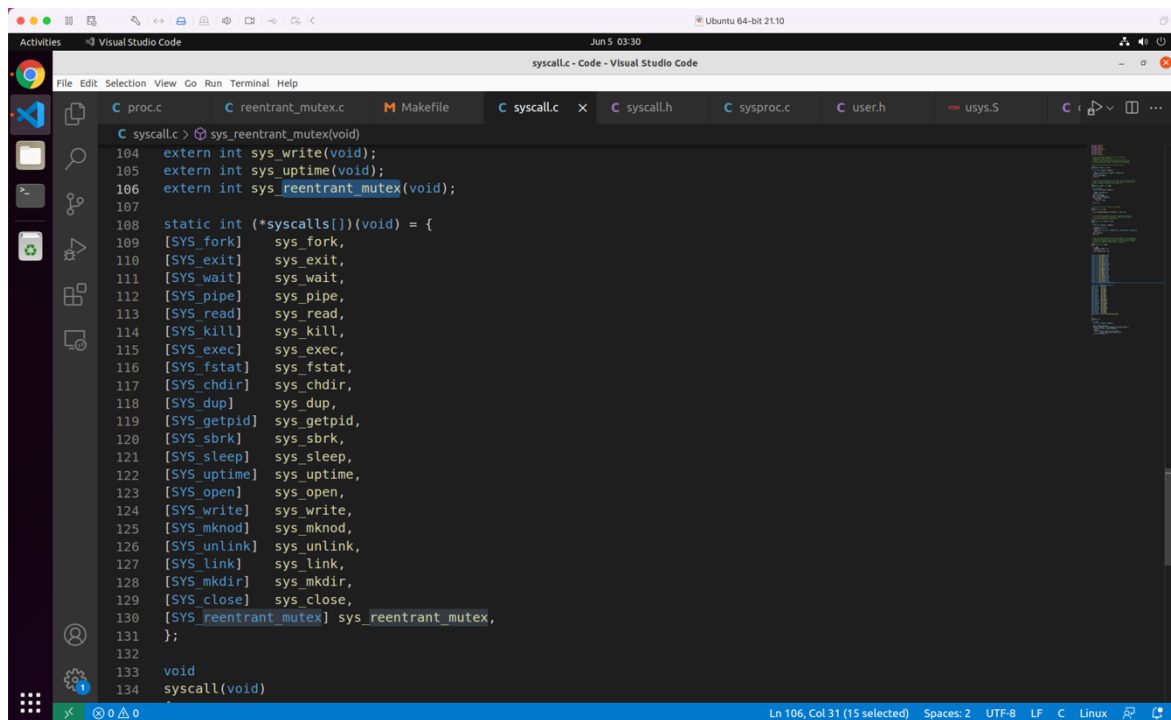
```
1  reentrant_mutex.c > main(int, char *[])
2  1 #include "types.h"
3  2 #include "stat.h"
4  3 #include "user.h"
5
6  int main(int argc, char *argv[])
7  {
8      int count = atoi(argv[1]);
9
10     printf(1, "acquire() count: %d\n", count);
11     reentrant_mutex(count);
12     exit();
13 }
```

تغييرات فايل makefile



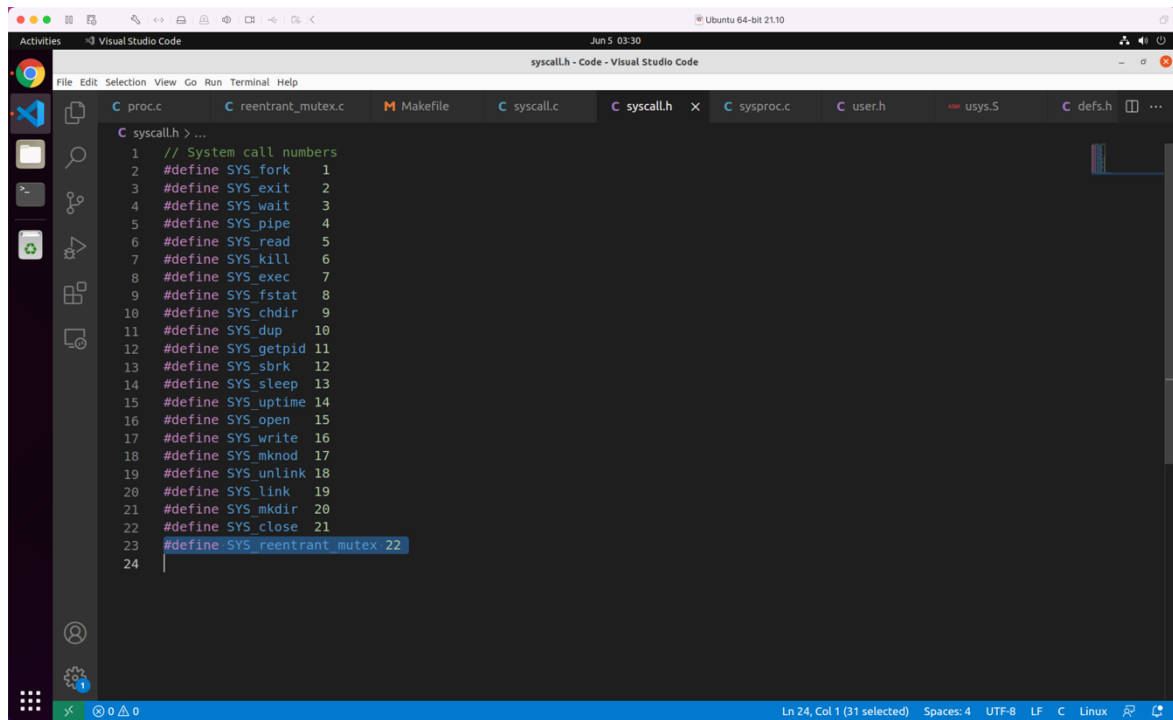
```
243 @echo "**** Now run 'gdb'." 1>62
244 $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUDEBUG)
245
246 # CUT HERE
247 # prepare dist for students
248 # after running make dist, probably want to
249 # rename it to rev0 or rev1 or so on and then
250 # check in that version.
251
252 EXTRA=\
253 mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
254 ln.c ls.c sort string.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
255 printf.c umalloc.c\
256 README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
257 .gdbinit.tmpl gdbutil\
258 reentrant_mutex
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
270
271 dist-test:
272 rm -rf dist
273 make dist
```

تغييرات فايل syscall.c



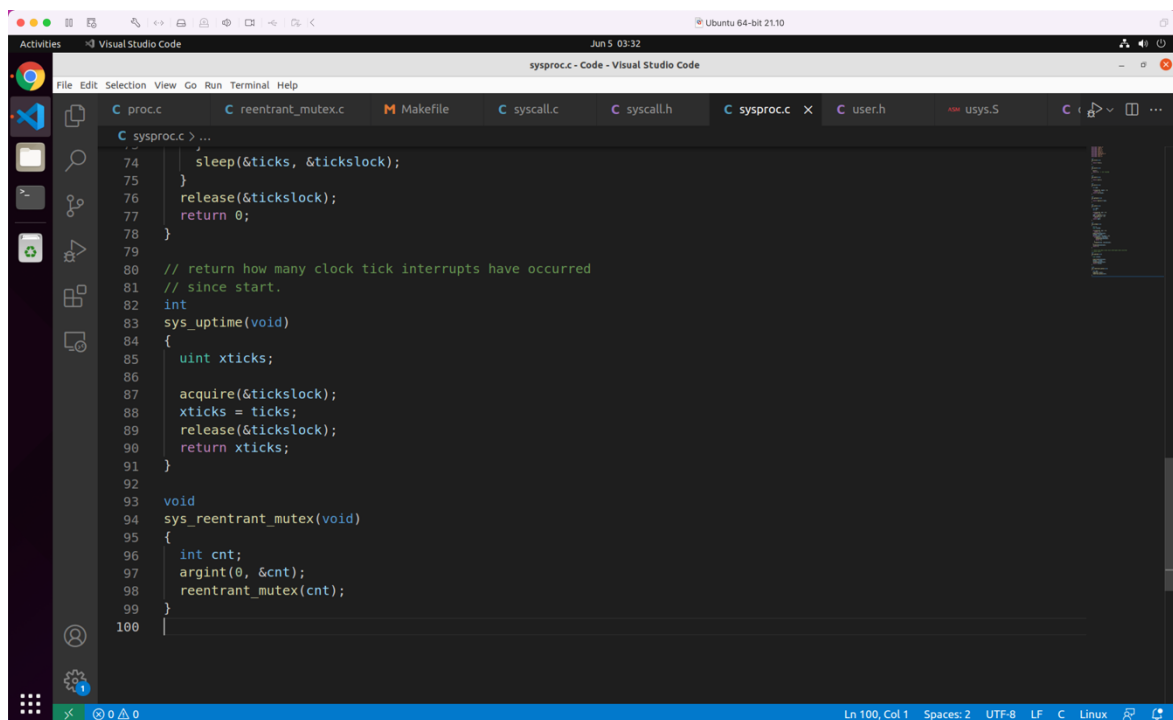
```
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_reentrant_mutex(void);
107
108 static int (*syscalls[])(void) = {
109 [SYS_fork] sys_fork,
110 [SYS_exit] sys_exit,
111 [SYS_wait] sys_wait,
112 [SYS_pipe] sys_pipe,
113 [SYS_read] sys_read,
114 [SYS_kill] sys_kill,
115 [SYS_exec] sys_exec,
116 [SYS_fstat] sys_fstat,
117 [SYS_chdir] sys_chdir,
118 [SYS_dup] sys_dup,
119 [SYS_getpid] sys_getpid,
120 [SYS_sbrk] sys_sbrk,
121 [SYS_sleep] sys_sleep,
122 [SYS_uptime] sys_uptime,
123 [SYS_open] sys_open,
124 [SYS_write] sys_write,
125 [SYS_mknod] sys_mknod,
126 [SYS_unlink] sys_unlink,
127 [SYS_link] sys_link,
128 [SYS_mkdir] sys_mkdir,
129 [SYS_close] sys_close,
130 [SYS_reentrant_mutex] sys_reentrant_mutex,
131 };
132
133 void
134 syscall(void)
```


تغییرات فایل syscall.h



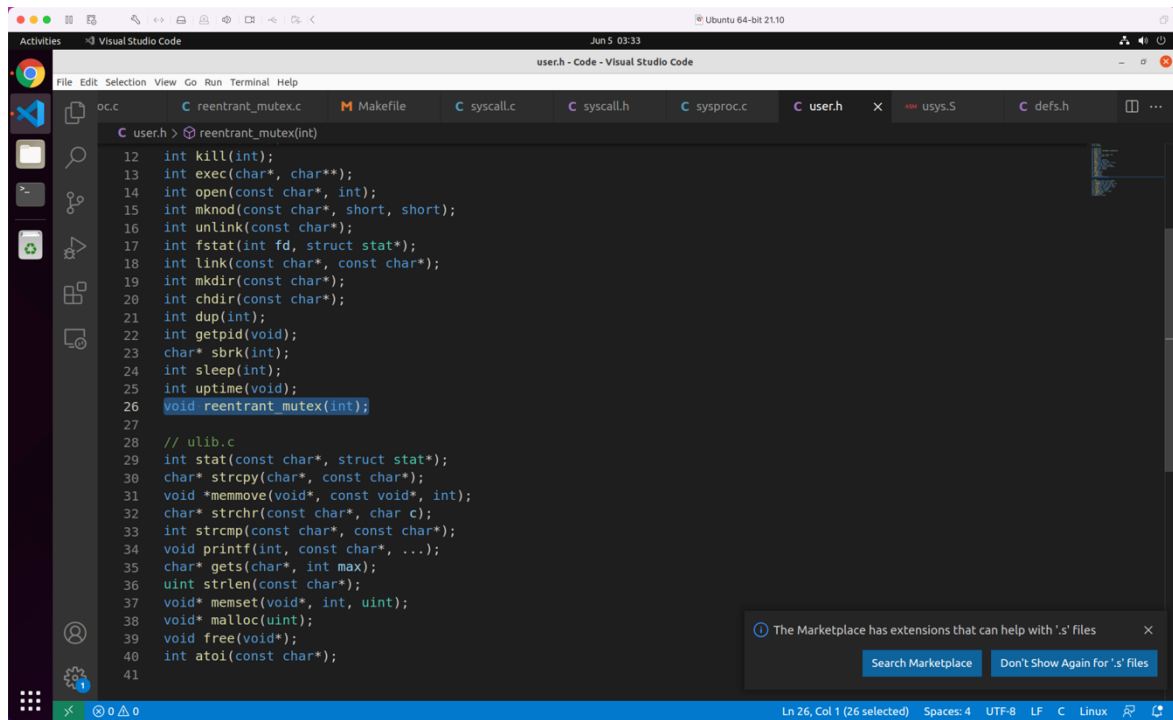
```
1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_reentrant_mutex 22
24
```

تغییرات فایل sysproc.c



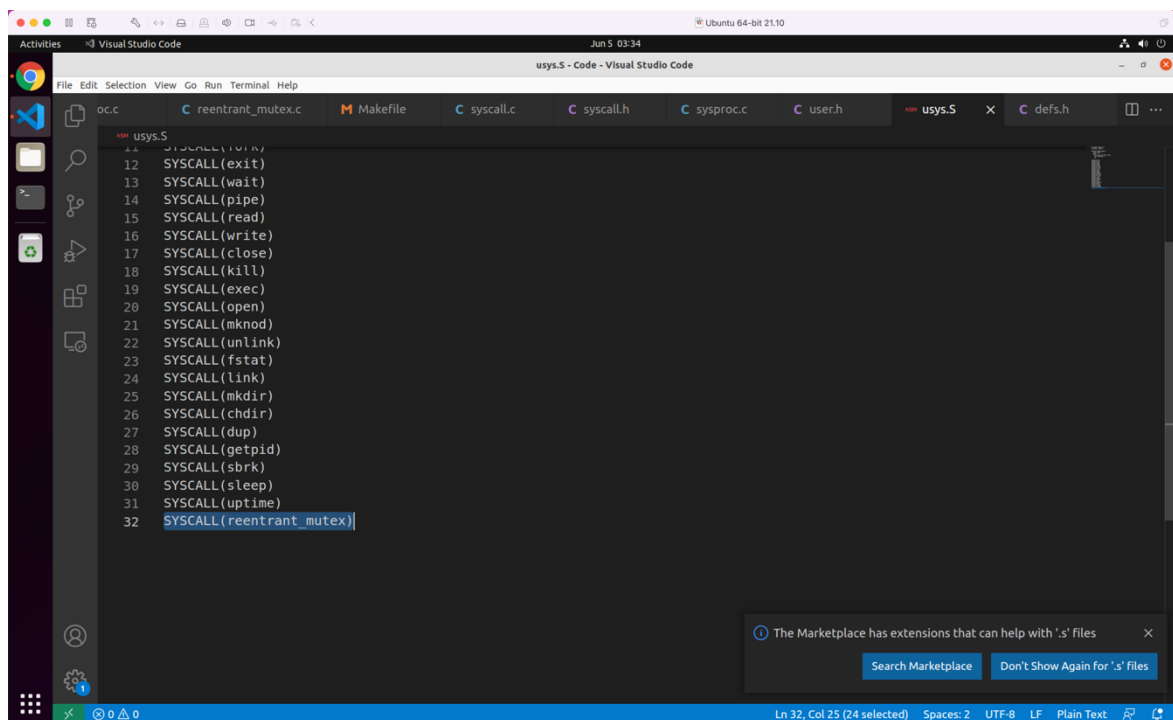
```
74 sleep(&ticks, &tickslock);
75 }
76 release(&tickslock);
77 return 0;
78 }
79
80 // return how many clock tick interrupts have occurred
81 // since start.
82 int
83 sys_uptime(void)
84 {
85     uint xticks;
86
87     acquire(&tickslock);
88     xticks = ticks;
89     release(&tickslock);
90     return xticks;
91 }
92
93 void
94 sys_reentrant_mutex(void)
95 {
96     int cnt;
97     argint(0, &cnt);
98     reentrant_mutex(cnt);
99 }
100
```

تغییرات فایل user.h

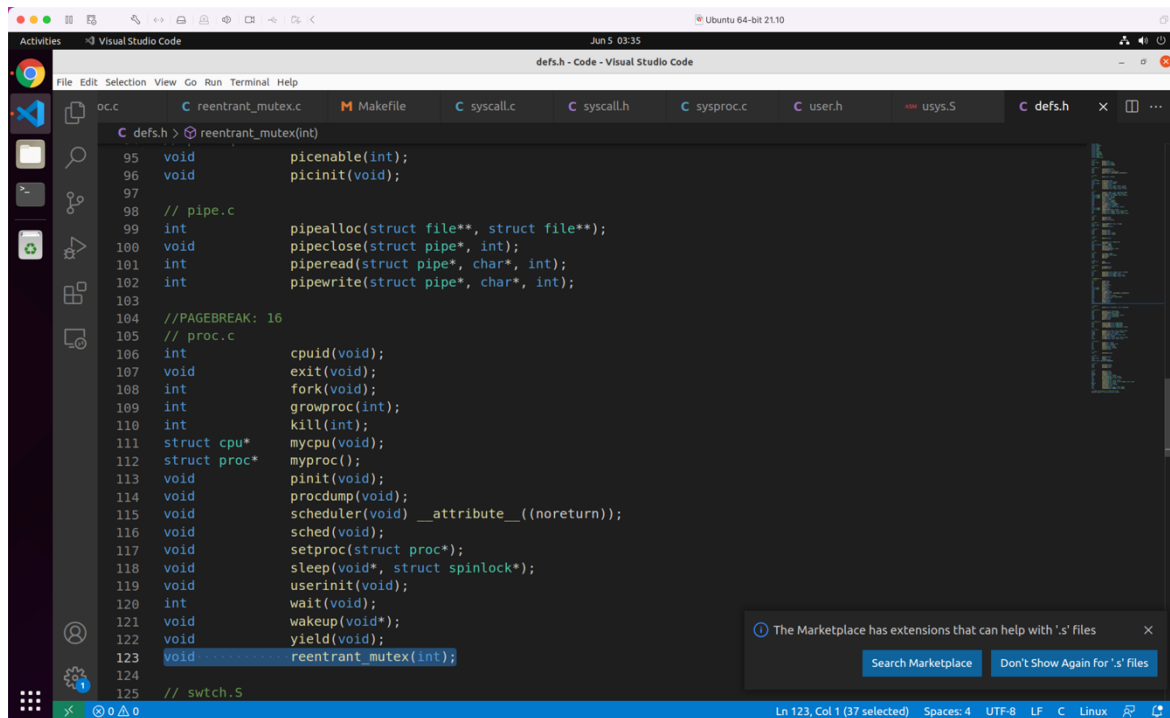


```
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 void reentrant_mutex(int);
27
28 // ulib.c
29 int stat(const char*, struct stat*);
30 char* strcpy(char*, const char*);
31 void *memmove(void*, const void*, int);
32 char* strchr(const char*, char c);
33 int strcmp(const char*, const char*);
34 void printf(int, const char*, ...);
35 char* gets(char*, int max);
36 uint strlen(const char*);
37 void* memset(void*, int, uint);
38 void* malloc(uint);
39 void free(void*);
40 int atoi(const char*);
41
```

خط اضافه شده به usys.S

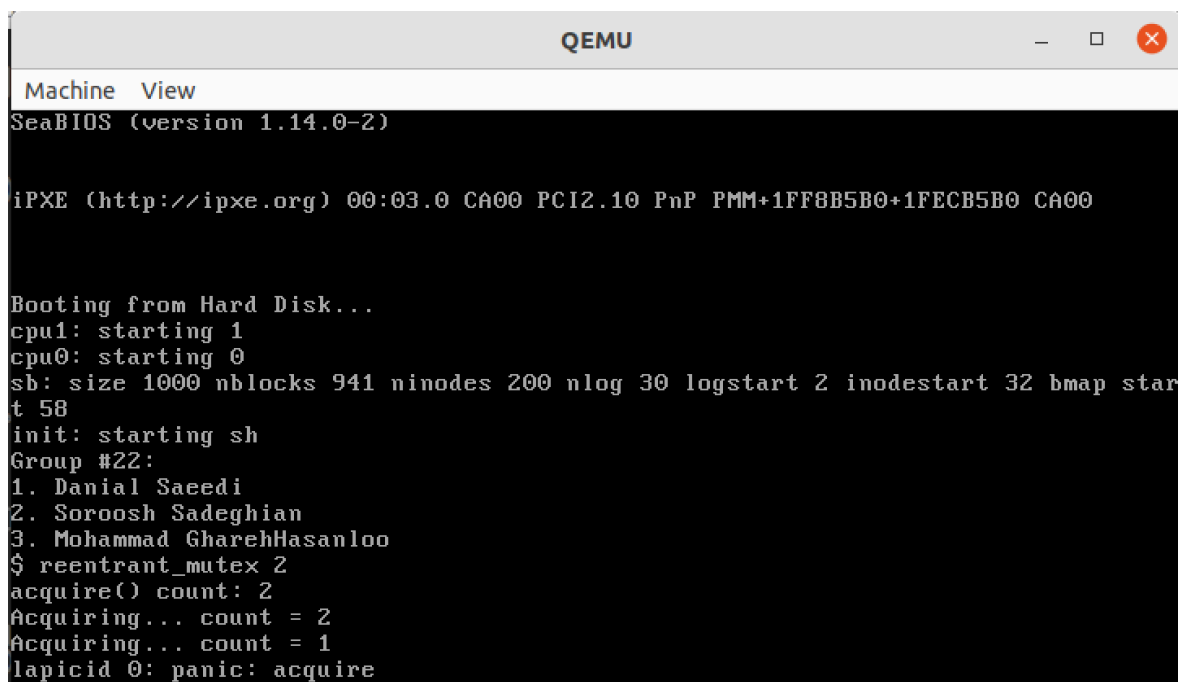


```
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(reentrant_mutex)
```



```
123 void reentrant_mutex(int);
```

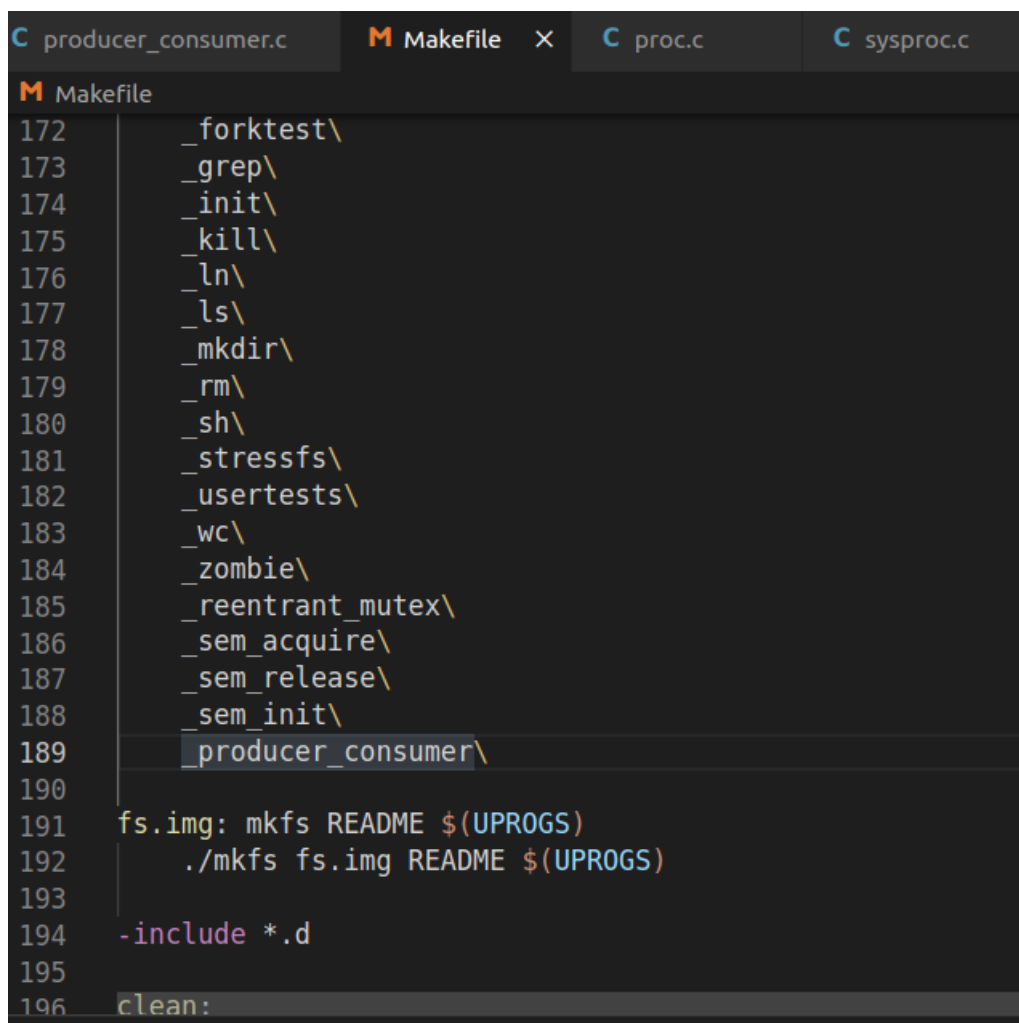
خروجی:



```
Machine View
SeaBIOS (version 1.14.0-2)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B5B0+1FECB5B0 CA00

Booting from Hard Disk...
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
Group #22:
1. Danial Saeedi
2. Soroosh Sadeghian
3. Mohammad GharehHasanloo
$ reentrant_mutex 2
acquire() count: 2
Acquiring... count = 2
Acquiring... count = 1
lapicid 0: panic: acquire
```



The screenshot shows a code editor with four tabs: `producer_consumer.c`, `Makefile`, `proc.c`, and `sysproc.c`. The `Makefile` tab is active, displaying a list of targets from line 172 to 196. The targets include various system utilities like `_forktest\`, `_grep\`, `_init\`, `_kill\`, `_ln\`, `_ls\`, `_mkdir\`, `_rm\`, `_sh\`, `_stressfs\`, `_usertests\`, `_wc\`, `_zombie\`, `_reentrant_mutex\`, `_sem_acquire\`, `_sem_release\`, `_sem_init\`, and `_producer_consumer\`. Line 189 has `_producer_consumer\` highlighted. Lines 191 and 192 show build instructions for `fs.img`. Line 194 shows `-include *.d`. Line 196 shows `clean:`.

```
172 _forktest\  
173 _grep\  
174 _init\  
175 _kill\  
176 _ln\  
177 _ls\  
178 _mkdir\  
179 _rm\  
180 _sh\  
181 _stressfs\  
182 _usertests\  
183 _wc\  
184 _zombie\  
185 _reentrant_mutex\  
186 _sem_acquire\  
187 _sem_release\  
188 _sem_init\  
189 _producer_consumer\  
190  
191 fs.img: mkfs README $(UPROGS)  
192 ./mkfs fs.img README $(UPROGS)  
193  
194 -include *.d  
195  
196 clean:
```

```

C sem_init.c > main(int, char *[])
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4  #include "fs.h"
5  #include "fcntl.h"
6
7  int main(int argc, char *argv[])
8  {
9      int arg1, arg2;
10
11     if(argc > 2) {
12         arg1 = atoi(argv[1]);
13         arg2 = atoi(argv[2]);
14     }
15     else {
16         printf(1, "Invalid inputs for sem_init\n");
17         exit();
18     }
19
20     sem_init(arg1, arg2);
21
22     exit();
23 }

```

```
producer_consumer.c  C proc.c  C sysproc.c  C syscall.h  C syscall.c  C sem_release.c  C sem_acquire.c  ASM usys.S  X
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(reentrant_mutex)
33  SYSCALL(sem_acquire)
34  SYSCALL(sem_release)
35  SYSCALL(sem_init)
```

```
cer_consumer.c  C proc.c  C sysproc.c  C syscall.h  C syscall.c  C sem_release.c  C sem_acquire.c  C user.h  X
h > ...
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
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);
void reentrant_mutex(int);
int sem_acquire(int);
int sem_release(int);
int sem_init(int, int);
```

```
producer_consumer.c  C proc.c  C sysproc.c  C syscall.h  C syscall.c  C sem_release.c  C sem_acquire.c  X
sem_acquire.c > main(int, char *[])
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4  #include "fs.h"
5  #include "fcntl.h"
6
7  int main(int argc, char *argv[])
8  {
9      int arg1;
10
11      if(argc > 1) {
12          arg1 = atoi(argv[1]);
13      }
14      else {
15          printf(1, "Invalid inputs for sem_acquire\n");
16          exit();
17      }
18
19      sem_acquire(arg1);
20
21      exit();
22  }
```

```
producer_consumer.c  proc.c  sysproc.c  syscall.h  syscall.c  sem_release.c X
C sem_release.c > main(int, char *[])
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4  #include "fs.h"
5  #include "fcntl.h"
6
7  int main(int argc, char *argv[])
8  {
9      int arg1;
10
11     if(argc > 1) {
12         arg1 = atoi(argv[1]);
13     }
14     else {
15         printf(1, "Invalid inputs for sem_release\n");
16         exit();
17     }
18
19
20     sem_release(arg1);
21
22     exit();
23 }
```

```
er_consumer.c  C proc.c  C sysproc.c  C syscall.h  C syscall.c X
l.c > [0] syscalls
[SYS_isstat] sys_isstat,
[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_reentrant_mutex] sys_reentrant_mutex,
[SYS_sem_acquire] sys_sem_acquire,
[SYS_sem_release] sys_sem_release,
[SYS_sem_init] sys_sem_init,
];
```

```
producer_consumer.c  C proc.c  C sysproc.c  C syscall.h  X
C syscall.h > SYS_reentrant_mutex
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_reentrant_mutex 22
24 #define SYS_sem_acquire 23
25 #define SYS_sem_release 24
26 #define SYS_sem_init 25
27
```



```
C producer_consumer.c  C proc.c  C sysproc.c X
C sysproc.c > sys_sem_release(void)
99     }
100
101     int sys_sem_init(void)
102     {
103         int i, v;
104
105         if (argint(0, &i) < 0)
106             return -1;
107
108         if (argint(1, &v) < 0)
109             return -1;
110
111         return sem_init(i, v);
112     }
113
114     int sys_sem_acquire(void)
115     {
116         int i;
117
118         if (argint(0, &i) < 0)
119             return -1;
120
121         return sem_acquire(i);
122     }
123
124     int sys_sem_release(void)
125     {
126         int i;
127
128         if (argint(0, &i) < 0)
129             return -1;
130
131         return sem_release(i);
132     }
133
```

```
C producer_consumer.c  C proc.c  X
C proc.c > sem_release(int)
550 }
551
552 #define MAXPROC 100
553
554 typedef struct
555 {
556     int value;
557     struct proc* list[MAXPROC];
558     int last;
559 }semaphore;
560
561 semaphore sems[6];
562
563 void sem_sleep(struct proc *p1)
564 {
565     acquire(&ptable.lock);
566     p1->state = SLEEPING;
567     sched();
568     release(&ptable.lock);
569 }
570
571 void sem_wakeup(struct proc *p1)
572 {
573     acquire(&ptable.lock);
574     p1->state = RUNNABLE;
575     release(&ptable.lock);
576 }
577
578 int sem_init(int i , int v)
579 {
580     sems[i].value = v;
581     sems[i].last = 0;
582     return 0;
583 }
```

```
C producer_consumer.c  C proc.c  X
C proc.c > sem_acquire(int)
582     return 0;
583 }
584
585 int sem_acquire(int i)
586 {
587     if(sems[i].value <= 0)
588     {
589         struct proc* p = myproc();
590         sems[i].list[sems[i].last] = p;
591         sems[i].last++;
592         sem_sleep(p);
593     }
594     else
595         sems[i].value--;
596
597     return 0;
598 }
599
600 int sem_release(int i)
601 {
602     if(sems[i].last)
603     {
604         sems[i].last--;
605         struct proc* p = sems[i].list[sems[i].last];
606         sem_wakeup(p);
607     }
608     else
609         sems[i].value++;
610
611     return 0;
612 }
```

C producer_consumer.c X

C producer_consumer.c > [?] buf

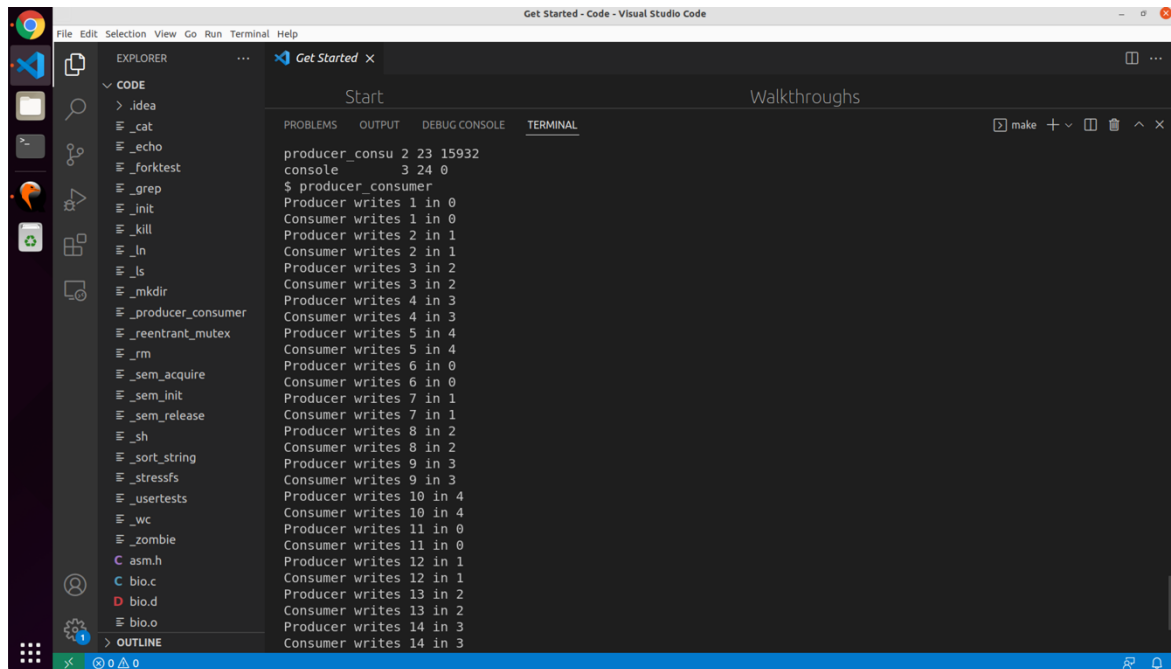
```
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4  #include "fcntl.h"
5
6
7  #define FULL 1
8  #define EMPTY 0
9  #define MUTEX 2
10
11 #define BUFF_SIZE 5
12 #define ITER_NUM 20
13
14 int buf[BUFF_SIZE];
15 int w_index = 0;
16 int r_index = 0;
17
18
19 void
20 producer()
21 {
22
23     for(int i = 1; i<ITER_NUM + 1; i++) {
24         sem_acquire(EMPTY);
25         sem_acquire(MUTEX);
26
27         printf(1, "Producer writes %d in %d\n", i, w_index % BUFF_SIZE);
28
29         sem_release(MUTEX);
30         sem_release(FULL);
31         w_index++;
32     }
33 }
```

```

C producer_consumer.c ×
C producer_consumer.c > ...
32 }
33 }
34
35 void
36 consumer()
37 {
38     for(int i = 1; i<ITER_NUM + 1; i++) {
39         sem_acquire(FULL);
40         sem_acquire(MUTEX);
41
42         printf(1, "Consumer writes %d in %d\n", i, r_index % BUFF_SIZE);
43
44         sem_release(MUTEX);
45         sem_release(EMPTY);
46         r_index++;
47     }
48 }
49
50 int
51 main(int argc, char *argv[])
52 {
53     sem_init(MUTEX, 1);
54     sem_init(FULL, BUFF_SIZE);
55     sem_init(EMPTY, BUFF_SIZE);
56
57     if (fork() == 0) producer();
58     else consumer();
59
60     wait();
61
62     exit();
63 }

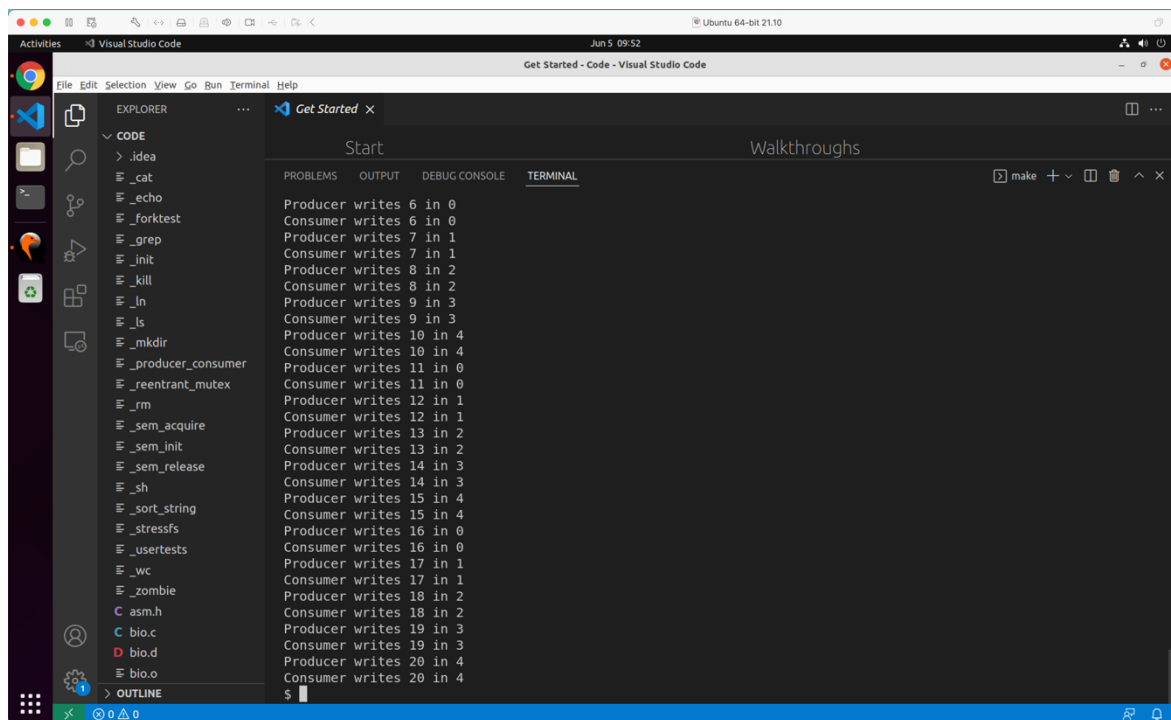
```

خروجی شبیه سازی:



```
Get Started - Code - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
CODE
  .idea
  _cat
  _echo
  _forktest
  _grep
  _init
  _kill
  _ln
  _ls
  _mkdir
  _producer_consumer
  _reentrant_mutex
  _rm
  _sem_acquire
  _sem_init
  _sem_release
  _sh
  _sort_string
  _stressfs
  _usertests
  _wc
  _zombie
  asm.h
  bio.c
  bio.d
  bio.o
OUTLINE
0 0 0

Start
Walkthroughs
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
$ make
producer_consu 2 23 15932
console 3 24 0
$ producer_consumer
Producer writes 1 in 0
Consumer writes 1 in 0
Producer writes 2 in 1
Consumer writes 2 in 1
Producer writes 3 in 2
Consumer writes 3 in 2
Producer writes 4 in 3
Consumer writes 4 in 3
Producer writes 5 in 4
Consumer writes 5 in 4
Producer writes 6 in 0
Consumer writes 6 in 0
Producer writes 7 in 1
Consumer writes 7 in 1
Producer writes 8 in 2
Consumer writes 8 in 2
Producer writes 9 in 3
Consumer writes 9 in 3
Producer writes 10 in 4
Consumer writes 10 in 4
Producer writes 11 in 0
Consumer writes 11 in 0
Producer writes 12 in 1
Consumer writes 12 in 1
Producer writes 13 in 2
Consumer writes 13 in 2
Producer writes 14 in 3
Consumer writes 14 in 3
```



```
Visual Studio Code
Jun 5 09:52
Get Started - Code - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
CODE
  .idea
  _cat
  _echo
  _forktest
  _grep
  _init
  _kill
  _ln
  _ls
  _mkdir
  _producer_consumer
  _reentrant_mutex
  _rm
  _sem_acquire
  _sem_init
  _sem_release
  _sh
  _sort_string
  _stressfs
  _usertests
  _wc
  _zombie
  asm.h
  bio.c
  bio.d
  bio.o
OUTLINE
0 0 0

Start
Walkthroughs
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
$ make
Producer writes 6 in 0
Consumer writes 6 in 0
Producer writes 7 in 1
Consumer writes 7 in 1
Producer writes 8 in 2
Consumer writes 8 in 2
Producer writes 9 in 3
Consumer writes 9 in 3
Producer writes 10 in 4
Consumer writes 10 in 4
Producer writes 11 in 0
Consumer writes 11 in 0
Producer writes 12 in 1
Consumer writes 12 in 1
Producer writes 13 in 2
Consumer writes 13 in 2
Producer writes 14 in 3
Consumer writes 14 in 3
Producer writes 15 in 4
Consumer writes 15 in 4
Producer writes 16 in 0
Consumer writes 16 in 0
Producer writes 17 in 1
Consumer writes 17 in 1
Producer writes 18 in 2
Consumer writes 18 in 2
Producer writes 19 in 3
Consumer writes 19 in 3
Producer writes 20 in 4
Consumer writes 20 in 4
$
```