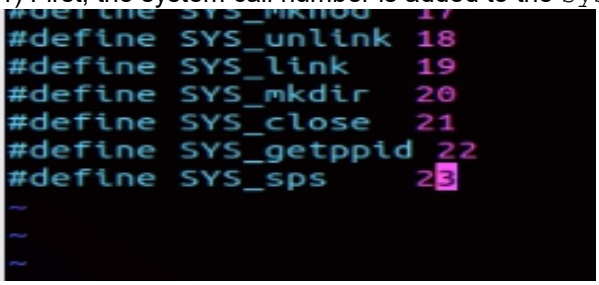
OS THEORY ASSIGNMENT - 3
NAME: ARPIT MOHAPATRA
ROLL NO: 1905383

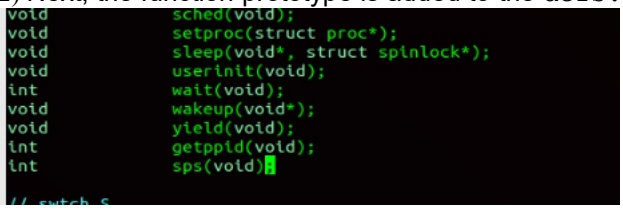
1. Create a system call called getppid() and create a command called "mytest" where you need to display the process-id along with parent process-id. (use the help of getpid)

Solution:

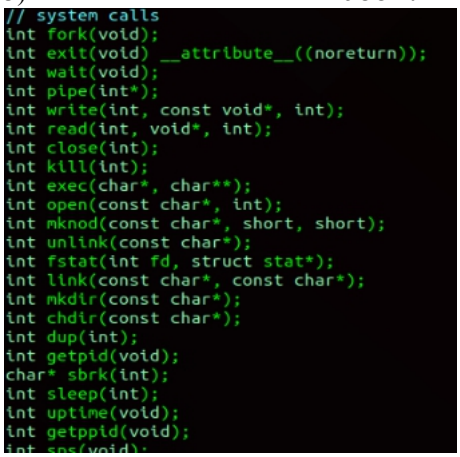
1) First, the system call number is added to the `syscall.h` header file(line 23)



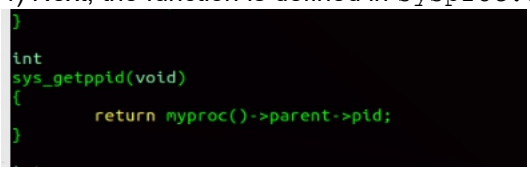
2) Next, the function prototype is added to the `defs.h` header file.(line 123)



3) The same is added to the `user.h` header file.(line 26)



4) Next, the function is defined in `sysproc.c` (line 93)



5) The system call is added to `usys.S` (line 32)

```
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(getppid)
SYSCALL(sps)
~
~
~
```

6) The function is also added to `syscall.c` (line 106 & 131)

```
extern int sys_uptime(void);
extern int sys_getppid(void);
extern int sys_sps(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,
[SYS_fstat]   sys_fstat,
[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_getppid] sys_getppid,
[SYS_sps]     sys_sps,
};
```

6) A new `mytest.c` file is created and the following is added, to use the new system call.

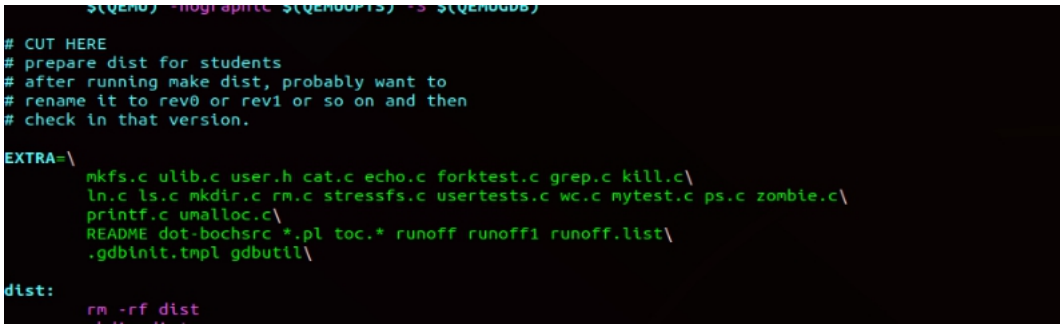
```
File  Edit  View  Search  Terminal  Help
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

int
main(int argc, char *argv[])
{
        printf(1, "Process id: %d\n", getpid());
        printf(1, "Parent process id: %d\n", getppid());
        exit();
}
~
~
~
~
```

8) Finally, the `Makefile` is modified to compile the `mytest.c` user program which uses our new system call. (line 183 & 254)

```
# Prevent deletion of
# that disk image chan
# details:
# http://www.gnu.org/s
.PRECIOUS: %.o

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



```
# 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 mytest.c ps.c zombie.c\
        printf.c umalloc.c\
        README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
        .gdbinit.tmpl gdbutil\

dist:
        rm -rf dist
```

9) Output:

```
balloc: write bitmap block at sector 58
qemu-system-i386 -nographic -drive file=f
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlo
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 13676
echo           2 4 12684
forktest       2 5 8112
grep           2 6 15552
init           2 7 13268
kill           2 8 12740
ln             2 9 12636
ls             2 10 14820
mkdir          2 11 12820
rm             2 12 12796
sh             2 13 23284
stressfs       2 14 13468
usertests      2 15 56396
wc             2 16 14216
mytest         2 17 12628
ps             2 18 12416
zombie         2 19 12460
console        3 20 0
$ mytest
Process id: 4
Parent process id: 2
$
```

2. Create a ps command that will display the following. You need to prepare a system call called sps(system processes) that will provide the following information. PID, PPID, Process name, process state then you try to display the following Your roll no, PID, PPID, Process name, process state, process creation time, size of process memory
Solution:
1. First, the system call number is added to the `syscall.h` header file(line 24)

```
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_getppid 22
#define SYS_sps    23
~
~
~
```

2. Next, the function prototype is added to the `defs.h` header file.(line 124)

```
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
int             getppid(void);
int             sps(void);
```

3. The same is added to the `user.h` header file.(line 27)

```
// system calls
int fork(void);
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);
int getppid(void);
int sps(void);
```

4. Next, the function is defined in `sysproc.c` (line 99)

```
        return myproc()->parent->pid;
}

int
sys_sps(void)
{
        return sps();
}
"sysproc.c" 103L, 1196C
```

5. Start time property is added to proc struct in `proc.h` (line 42)

```
// 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
  uint stime;                  // Start time
  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
"proc.h" 59L, 2310C
```

6. `sps()` function is added in `proc.c` with other changes to account for start time. (line 91 & 551)

```
86   return 0;
87
88 found:
89   p->state = EMBRYO;
90   p->pid = nextpid++;
91   p->stime = ticks;
92
93   release(&ptable.lock);
94
```

```
551 int sps(void)
552 {
553         struct proc *p;
554         char *states[] = {
555                 [UNUSED] "Unused",
556                 [EMBRYO] "Embroy",
557                 [SLEEPING] "Sleeping",
558                 [RUNNABLE] "Runnable",
559                 [RUNNING] "Running",
560                 [ZOMBIE] "Zombie",
561         };
562         acquire(&ptable.lock);
563         cprintf("ROLL\tPID\tPPID\tNAME\tSIZE\tSTART\tSTATE\n");
564         for(p = ptable.proc; p<&ptable.proc[NPROC]; p++) {
565                 if(p->state != UNUSED) {
566                         cprintf("1905383");
567                         cprintf("\t%d", p->pid);
568                         if(p->parent->pid/10 == 0) {
569                                 cprintf("\t%d", p->parent->pid);
570                         }
571                         else {
572                                 cprintf("\tNone");
573                         }
574                 }
575                 cprintf("\t%s", p->name);
576                 cprintf("\t%d", p->sz);
577                 cprintf("\t%d", p->stime);
578                 cprintf("\t%s", states[p->state]);
579                 cprintf("\n");
580         }
581         release(&ptable.lock);
582         return 23;
583 }
```

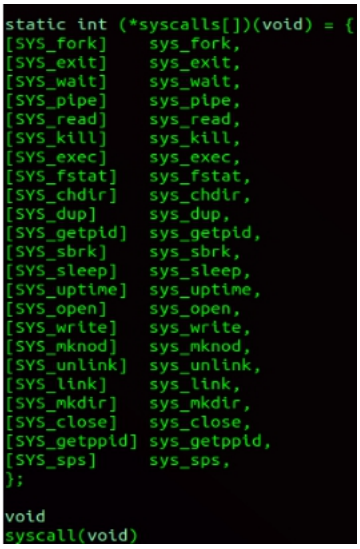
7. The system call is added to `usys.S` (line 33)

```
File  Edit  View  Search  Terminal  Help
#include "syscall.h"
#include "traps.h"

#define SYSCALL(name) \
  .globl name; \
  name: \
    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(getppid)
SYSCALL(sps)
```

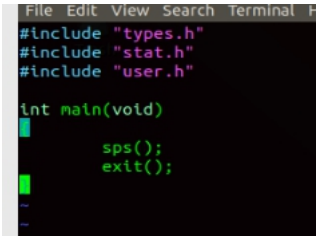
8. The function is also added to `syscall.c` (line 107 & 132)



```
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,
[SYS_fstat]   sys_fstat,
[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_getppid] sys_getppid,
[SYS_sps]     sys_sps,
};

void
syscall(void)
```

9. A new `ps.c` file is created and the following is added, to use the new system call.



```
File  Edit  View  Search  Terminal
#include "types.h"
#include "stat.h"
#include "user.h"

int main(void)
{
        sps();
        exit();
}
```

10. Finally, the `Makefile` is modified to compile the `ps.c` user program which uses our new system call. (line 184 & 254)

```
# Prevent deletion of
# that disk image chan
# details:
# http://www.gnu.org/s
.PRECIOUS: %.o

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

```
# 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 mytest.c ps.c zombie.c\
	printf.c umalloc.c\
	README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
	.gdbinit.tmpl gdbutil\

dist:
	rm -rf dist
```

11. Output:

```
350+1 records out
179620 bytes (180 kB, 175 KiB) copied, 0.00142921 s, 126 MB/s
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,form
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 13676
echo           2 4 12684
forktest       2 5 8112
grep           2 6 15552
init           2 7 13268
kill           2 8 12740
ln             2 9 12636
ls             2 10 14820
mkdir          2 11 12820
rm             2 12 12796
sh             2 13 23284
stressfs       2 14 13468
usertests      2 15 56396
wc             2 16 14216
mytest         2 17 12628
ps             2 18 12416
zombie         2 19 12460
console        3 20 0
$ ps
ROLL    PID     PPID    NAME    SIZE    START   STATE
1905383 1       None    init    12288   0       Sleeping
1905383 2       1       sh      16384   31      Sleeping
1905383 4       2       ps      12288   383     Running
                0       0       Unused
```