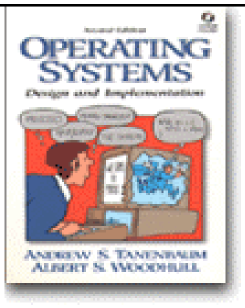
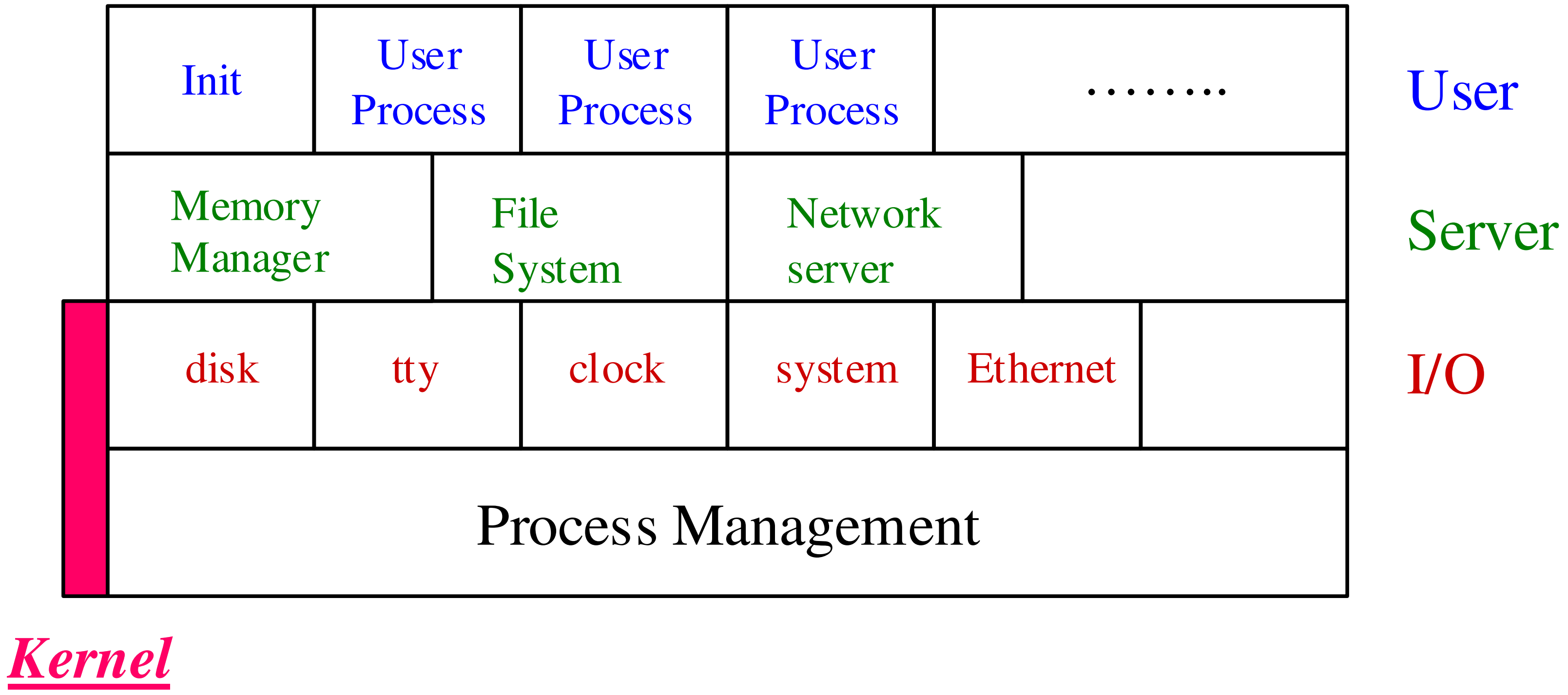
ECS 150 - Discussion section - 10/27 and 10/28

- Review of Minix Architecture
- Tracing a Minix System Call to the Kernel
- Changes to library code
- Changes to MM and Kernel
- Compilation

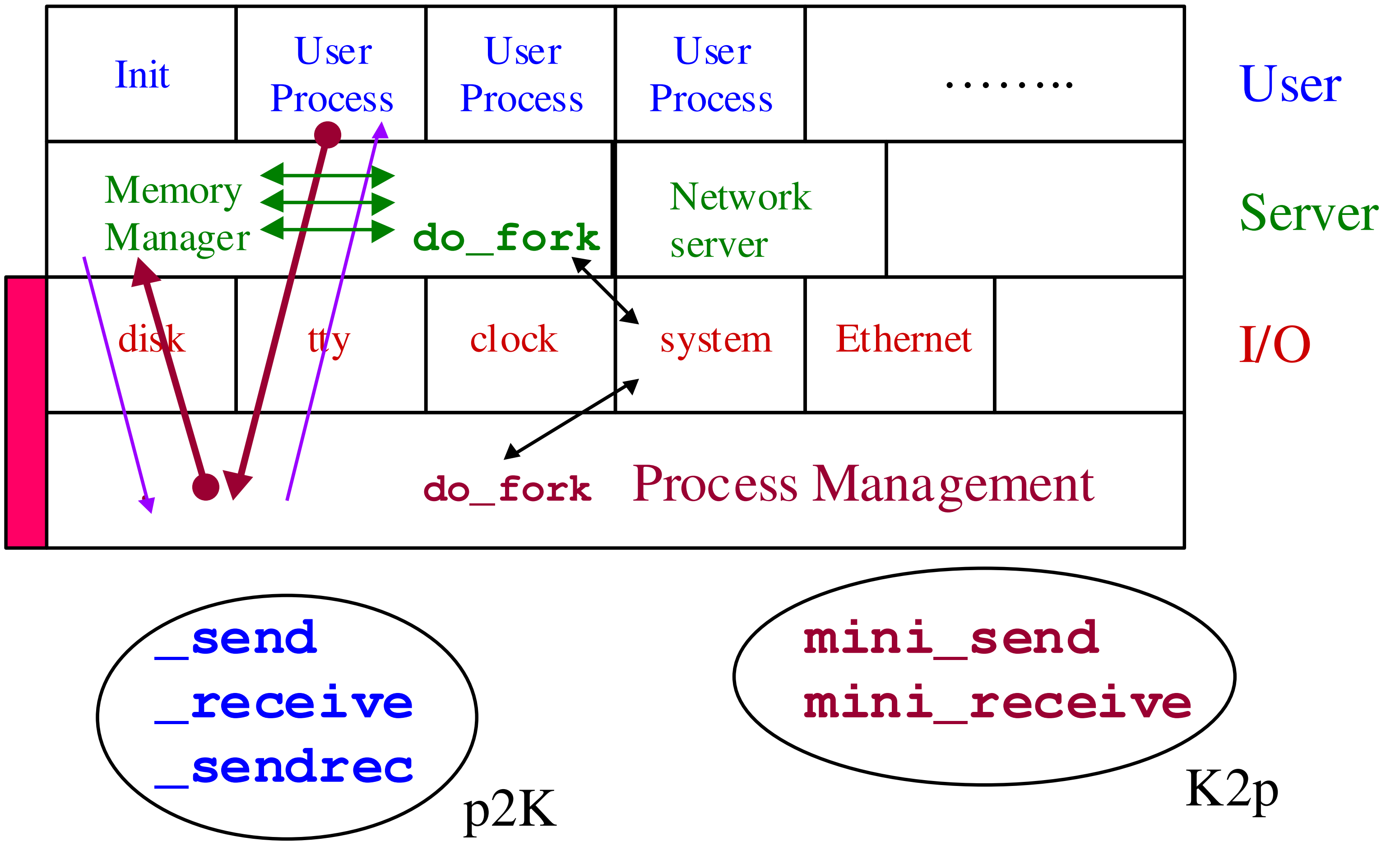


Minix OS Structure



Taken from: http://www.cs.ucdavis.edu/~wu/ecs150/ecs150_F2003_L001.ppt, written by Dr. S. Felix Wu.

Calling Diagram

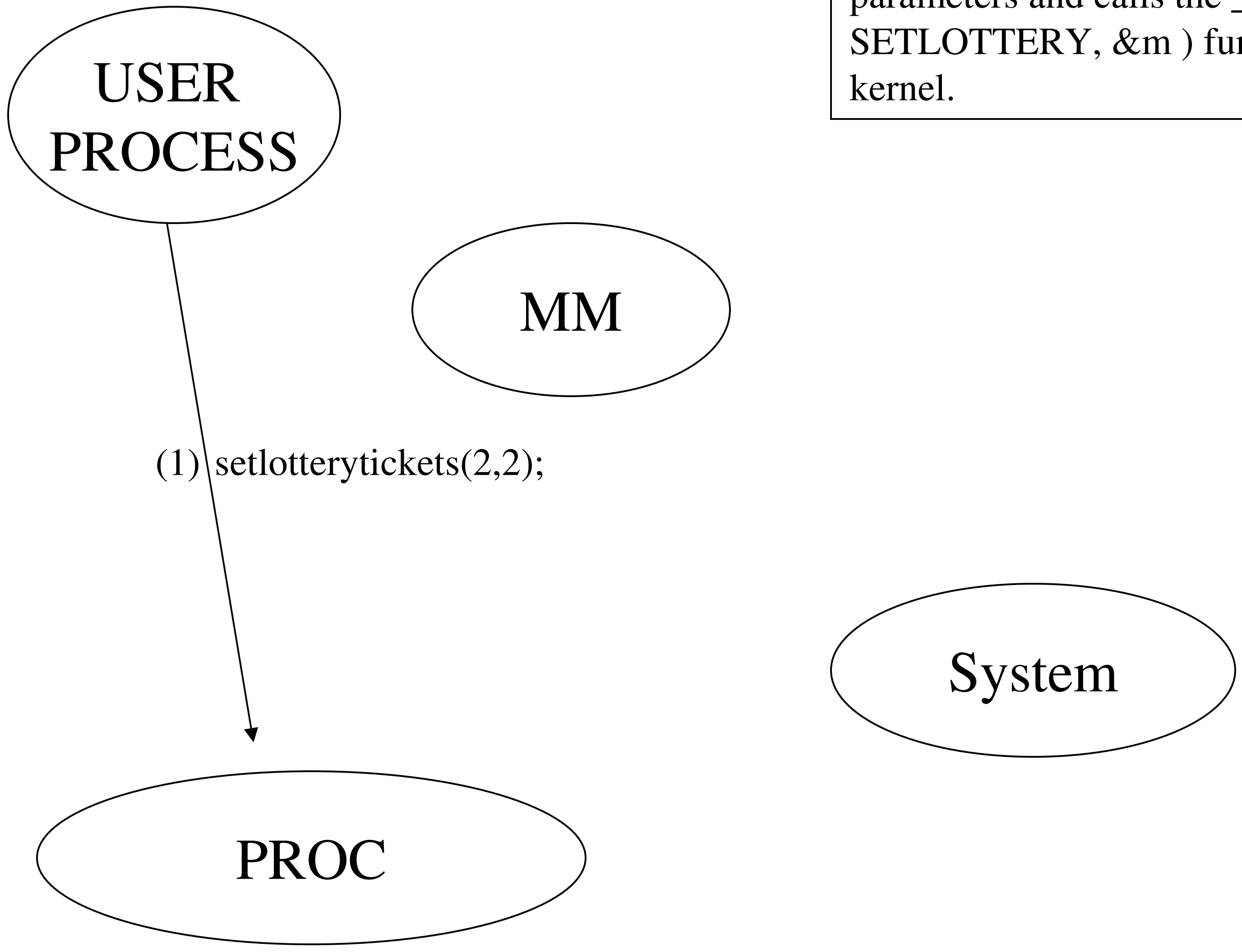


Taken from: http://www.cs.ucdavis.edu/~wu/ecs150/ecs150_F2003_L002.ppt, written by Dr. S. Felix Wu.

Overview of Syscall control flow

(1) /usr/include/setlottery.h:

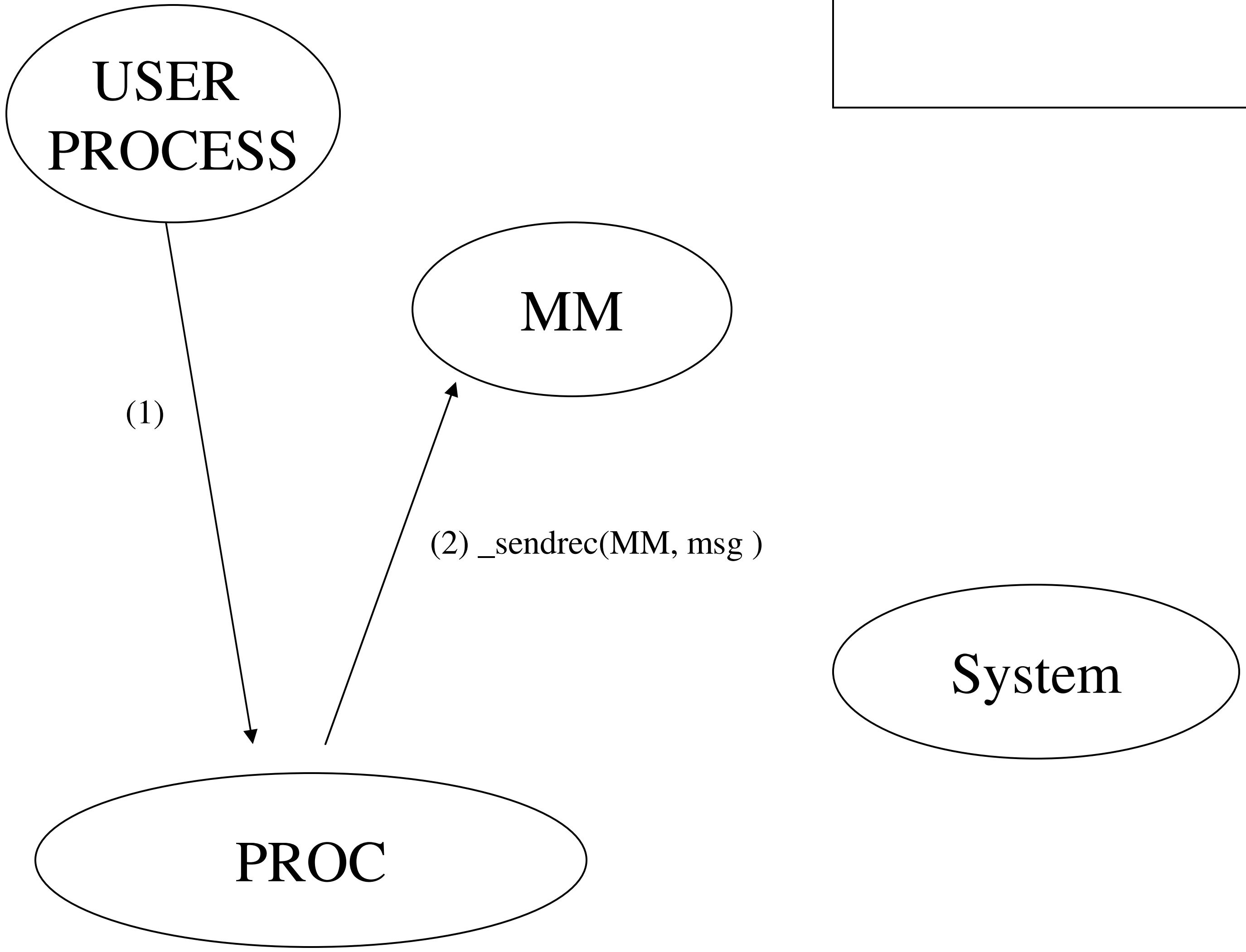
setlotterytickets() - library function call accessible from user space. Marshals parameters and calls the _syscall(MM, SETLOTTERY, &m) function trapping to kernel.



Overview of Syscall control flow

(1) setlottery.h: **setlotterytickets**() - traps to kernel

(2) /usr/src/lib/other/syscall.c: _syscall() - the syscall function determines the message is for the MM and passes control to the MM.



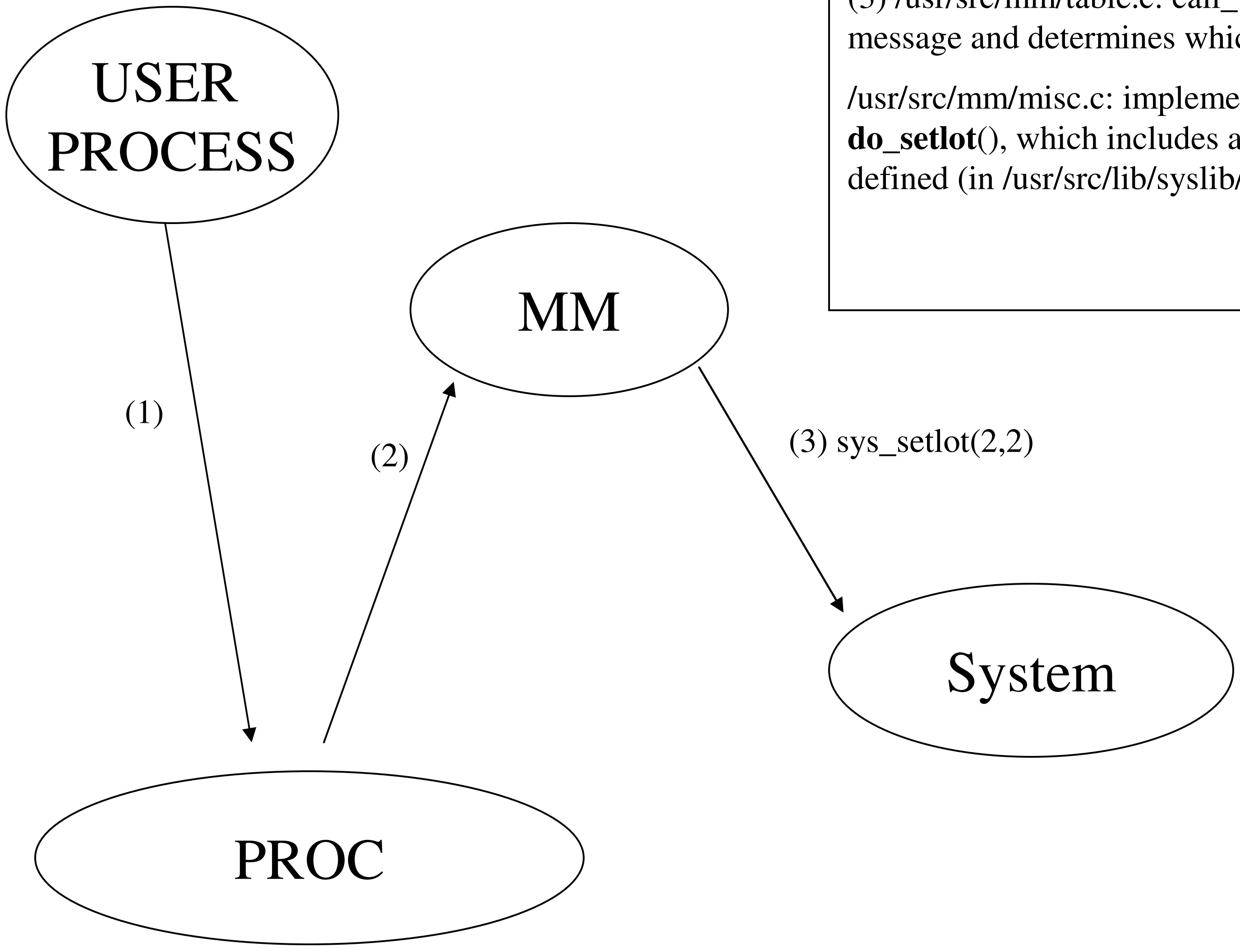
Overview of Syscall control flow

(1) setlottery.h: **setlotterytickets**() - traps to kernel

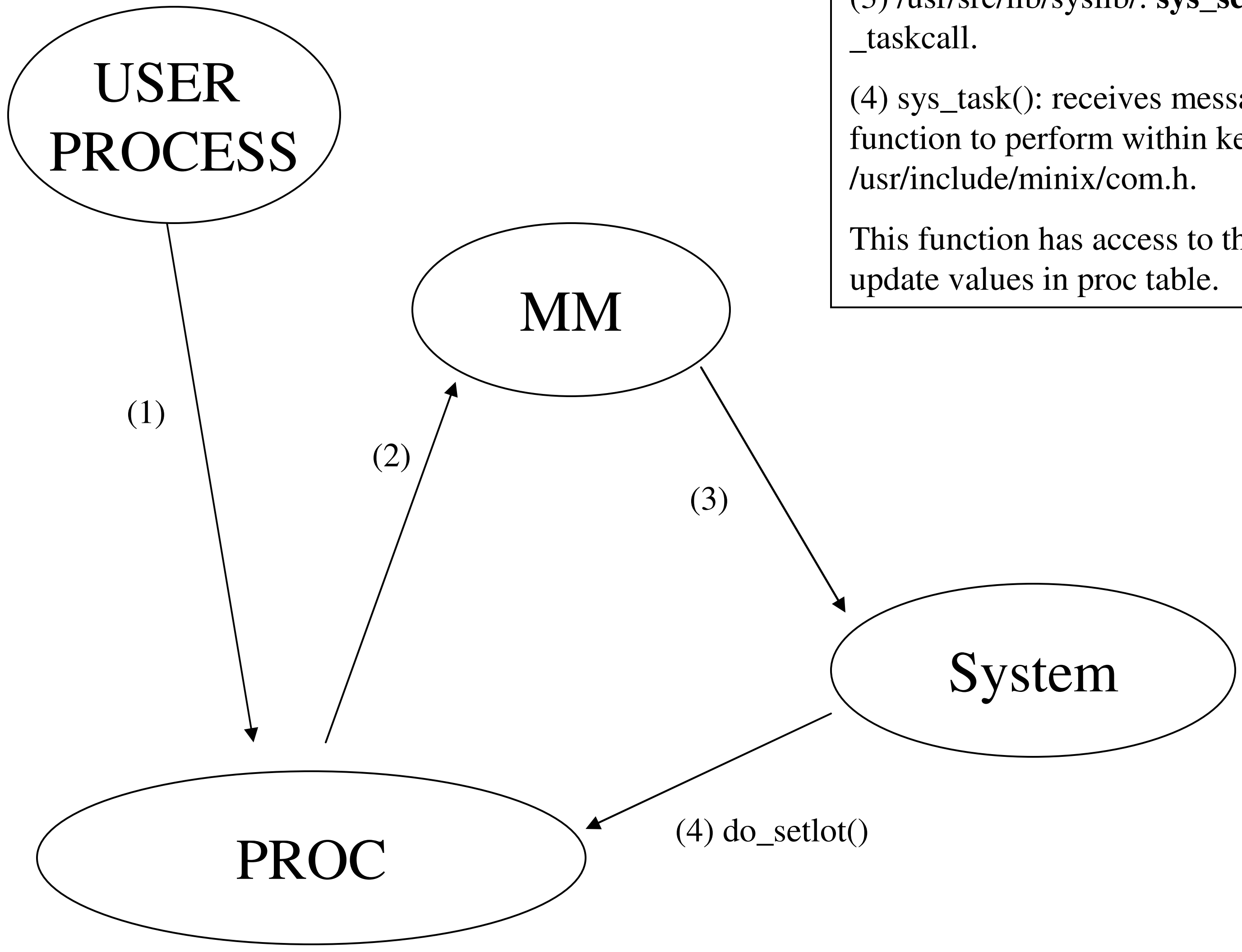
(2) /usr/src/lib/other/syscall.c: sendrec() directs message to MM

(3) /usr/src/mm/table.c: call_vec() function receives message and determines which function to execute.

/usr/src/mm/misc.c: implements function, e.g. **do_setlot**(), which includes a call to sys_sycall defined (in /usr/src/lib/syslib/**sys_setlot.c**.)



Overview of Syscall control flow



(1) setlottery.h: **setlotterytickets**() - traps to kernel

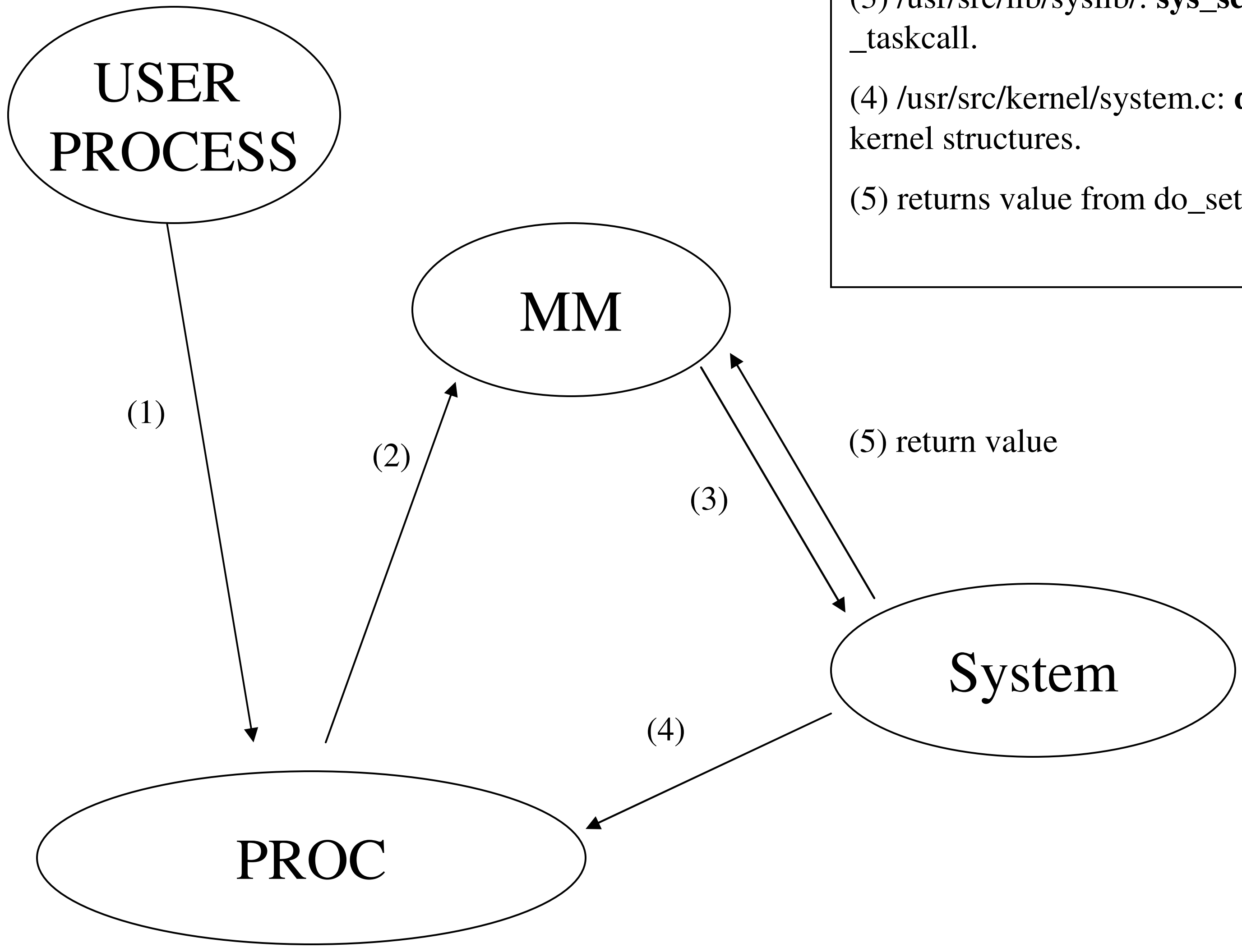
(2) /usr/src/lib/other/syscall.c: sendrec() directs message to MM

(3) /usr/src/lib/syslib/: **sys_setlot**(2,2) invokes _taskcall.

(4) sys_task(): receives message, determines which function to perform within kernel code, as defined in /usr/include/minix/com.h.

This function has access to the kernel globals and can update values in proc table.

Overview of Syscall control flow



(1) setlottery.h: **setlotterytickets**() - traps to kernel

(2) /usr/src/lib/other/syscall.c: sendrec() directs message to MM

(3) /usr/src/lib/syslib/: **sys_setlot**(2,2) invokes _taskcall.

(4) /usr/src/kernel/system.c: **do_setlot**() modifies kernel structures.

(5) returns value from do_setlot() function.

Overview of Syscall control flow

(1) setlottery.h: **setlotterytickets**() - traps to kernel

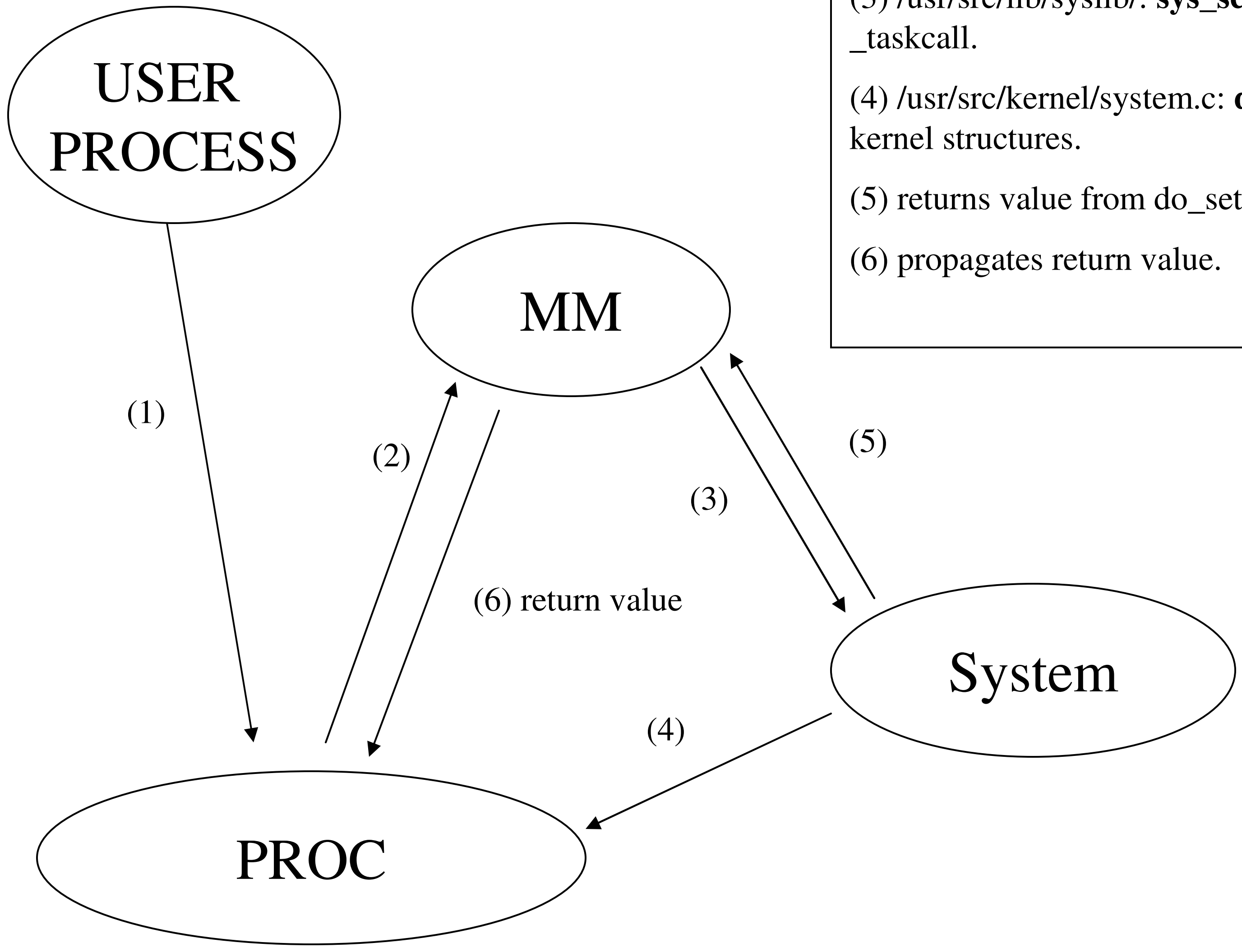
(2) /usr/src/lib/other/syscall.c: sendrec() directs message to MM

(3) /usr/src/lib/syslib/: **sys_setlot**(2,2) invokes _taskcall.

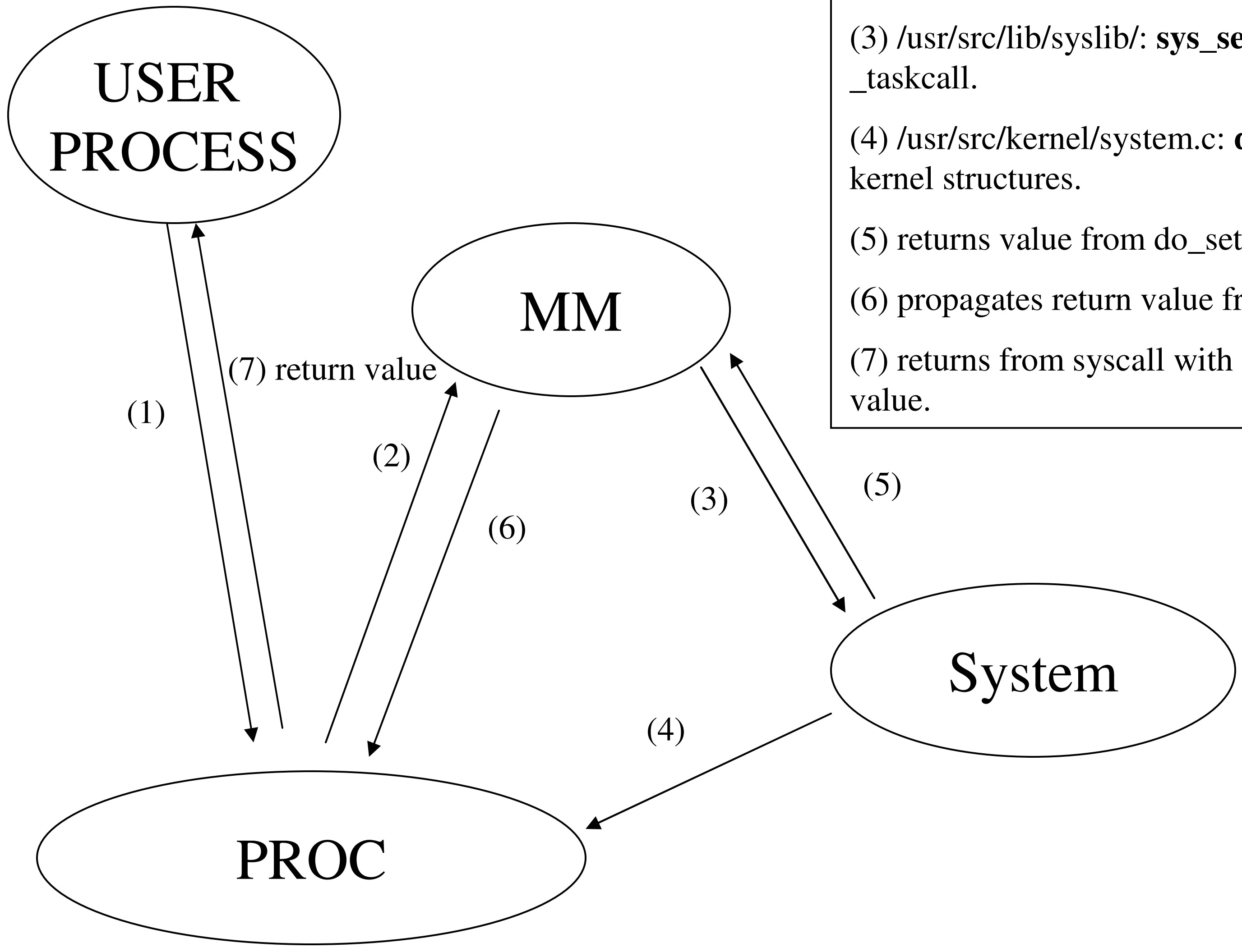
(4) /usr/src/kernel/system.c: **do_setlot**() modifies kernel structures.

(5) returns value from do_setlot() function.

(6) propagates return value.



Overview of Syscall control flow



(1) setlottery.h: **setlotterytickets()** - traps to kernel

(2) /usr/src/lib/other/syscall.c: sendrec() directs message to MM, which uses its version of **do_setlot()** to access the kernel.

(3) /usr/src/lib/syslib/: **sys_setlot(2,2)** invokes _taskcall.

(4) /usr/src/kernel/system.c: **do_setlot(m)** modifies kernel structures.

(5) returns value from do_setlot() function.

(6) propagates return value from do_setlot().

(7) returns from syscall with the propagated return value.

Files that need to be added/modified:

Library (requires library to be compiled):

/usr/src/lib/syslib/Makefile - add object file for sys_*yoursyscall*.c

/usr/src/lib/syslib/sys_*yoursyscall*.c - invoke _taskcall

Includes:

/usr/include/minix/com.h - set SETLOTTERY number

/usr/include/minix/callnr.h - set system call number

/usr/include/minix/syslib.h - add prototype for user accessible sys_syscall

/usr/include/unistd.h - add prototype for syscall

/usr/include/*YOUR_SYSCALL*.h - invoke _syscall

Files that need to be added/modified:

FS (compiled through /usr/src/tools):

/usr/src/fs/table.c - put entry in call vec

MM (compiled through /usr/src/tools):

/usr/src/mm/table.c - add entry to call vec for MM side do_setlottery

/usr/src/mm/misc.c - implement MM side do_setlottery

/usr/src/mm/proto.h - add prototype for do_setlottery

Kernel (compiled through /usr/src/tools):

/usr/src/kernel/proc.c - add code for lottery

/usr/src/kernel/proc.h - add a field for num_tickets

/usr/src/kernel/system.c - add kernel side do_setlottery

Changes to library - /usr/include/

Todo: edit unistd.h to include prototype for your function.

Example of adding int foo(int num, int num2):

Open `/usr/include/unistd.h`

Add `_PROTOTYPE( int foo, (int num, int num2) ); at the end of the`

`/* Function Prototypes*/ section (around line 126)`

The end of the `/* Function Prototypes */ section should look like:`

```
/* Function Prototypes. */
_PROTOTYPE( void _exit, (int _status)              );

/* ... omitted code ... */

_PROTOTYPE( int foo, (int num, int num2)                   );
```

Example adapted from: http://wwwcsif.cs.ucdavis.edu/~cs150/syscall.html, written by Sophie Engle.

Changes to library - /usr/include/

Todo: edit *yourlib*.h to include a small user library. Invokes _sys_call to trap to kernel.

Example of syscall available to user, int foo(int num, int num2):

```
#include <lib.h>
#define foo _foo
#include <unistd.h>

PUBLIC int foo( int num, int num2 )
{
          message m;

          m.m4_l1 = num;
          m.m4_l2 = num2;

          return( _syscall( MM, FOO, &m ) );
}
```

* The message datatype is defined in /usr/include/minix/type.h

Example adapted from: http://wwwcsif.cs.ucdavis.edu/~cs150/syscall.html, written by Sophie Engle.

Changes to library - /usr/src/lib/syslib

Todo: create sys_*yoursyscall*.c within this directory and edit Makefile to compile this file (next slide).

Example of sys_fork (sys_fork.c):

```
#include "syslib.h"

PUBLIC int sys_fork(parent, child, pid, child_base_or_shadow)
int parent;                                /* process doing the fork */
int child;                                 /* which proc has been created by the fork
*/
int pid;                          /* process id assigned by MM */
phys_clicks child_base_or_shadow;          /* position for child [VM386];
                                            * memory allocated for shadow [68000] */
{
/* A process has forked.  Tell the kernel. */

  message m;

  m.m1_i1 = parent;
  m.m1_i2 = child;
  m.m1_i3 = pid;
  m.m1_p1 = (char *) child_base_or_shadow;
  return(_taskcall(SYSTASK, SYS_FORK, &m));
}
```

- Pass parameters via message.
- Invokes _taskcall.
- Pass function name as 2nd parameter (defined in /usr/include/minix/com.h).

Changes to library - /usr/src/lib/syslib

Editing Makefile:

```
# Makefile for lib/syslib.

CFLAGS    = -O -D_MINIX -D_POSIX_SOURCE
CC1       = $(CC) $(CFLAGS) -c

LIBRARY   = ../libc.a
all:      $(LIBRARY)

OBJECTS   = \
          $(LIBRARY)(sys_abort.o) \
… code omitted ...
          $(LIBRARY)(taskcall.o) \

$(LIBRARY):          $(OBJECTS)
          aal cr $@ *.o
          rm *.o

$(LIBRARY)(sys_abort.o):
          sys_abort.c
          $(CC1) sys_abort.c

… code omitted ...

$(LIBRARY)(taskcall.o):
          taskcall.c
          $(CC1) taskcall.c
```

Todo: add lines to Makefile so that the new syscall will be added to libc.a.

(1) Add object to compile into library

(2) Add instructions as to how to compile object

Changes to library - /usr/include/minix

Todo: edit com.h to add an index into sys_task() function (/usr/src/kernel/systemc.)

```
/* System calls. */
#define SEND                    1      /* function code for sending messages */
#define RECEIVE                 2      /* function code for receiving messages */
#define BOTH                    3      /* function code for SEND + RECEIVE */
#define ANY   (NR_PROCS+100)  /* receive(ANY, buf) accepts from any source */
… code omitted ...
#define SYSTASK           -2  /* internal functions */
#         define SYS_XIT        1       /* fcn code for sys_xit(parent, proc) */
#         define SYS_GETSP      2       /* fcn code for sys_sp(proc, &new_sp) */
… code omitted
#         define SYS_GETMAP    20       /* fcn code for sys_getmap(procno, map_ptr) */

… code omitted
```

Add #define to denote your SYS_*YOURSYS*

Changes to library - /usr/include/minix

Todo: edit syslib.h to include the prototype for the sys_*YOURSYSCALL*(int pid, int tickets) function.

```
/* Prototypes for system library functions. */
#ifndef _SYSLIB_H
#define _SYSLIB_H

/* Hide names to avoid name space pollution. */
#define sendrec              _sendrec
#define receive              _receive
#define send                 _send

/* Minix user+system library. */
_PROTOTYPE( void printk, (char *_fmt, ...)                                    );
_PROTOTYPE( int sendrec, (int _src_dest, message *_m_ptr)             );
_PROTOTYPE( int _taskcall, (int _who, int _syscallnr, message *_msgptr)        );

/* Minix system library. */
_PROTOTYPE( int receive, (int _src, message *_m_ptr)                          );
_PROTOTYPE( int send, (int _dest, message *_m_ptr)                            );

_PROTOTYPE( int sys_abort, (int _how, ...)                                    );
… code omitted ...
_PROTOTYPE( int sys_times, (int _proc, clock_t _ptr[5])                       );

#endif /* _SYSLIB_H */
```

Changes to library - /usr/include/minix

Todo: edit callnr.h to to give your system call a specific integer value.

```
#define NCALLS                 77      /* number of system calls allowed */

#define EXIT                    1
#define FORK                    2
… code omitted ...
#define PAUSE                  29
#define UTIME                  30
#define ACCESS                 33
#define SYNC                   36
#define KILL                   37
… code omitted ...
#define REBOOT                 76
```

Add #define to denote your *YOURSYSCALL*. Either select a value between 1 and NCALLS that is not in use, or increase NCALLS by 1 and #define as NCALLS - 1.

Changes to FS - /usr/src/fs/table.c

Todo: add an entry into call_vec table to handle new system call. If you used an unused system call number, ensure that the command is no_sys. Otherwise, add to end as outlined in syscall handout.

Go to the end of the `call_vec array`

Add the line `no_sys, /* 78 = FOO */ at the end of call_vec`

Save and exit

Our new `call_vec should look like this:`

```
PUBLIC _PROTOTYPE( int (*call_vec[]), (void) ) = {
          no_sys,   /* 0 = unused      */
          do_exit,  /* 1 = exit                  */

          /* ... omitted code ... */

          do_svrctl,          /* 77 = SVRCTL      */
          no_sys,   /* 78 = FOO                  */
};
```

Example adapted from: http://wwwcsif.cs.ucdavis.edu/~cs150/syscall.html, written by Sophie Engle.

Changes to MM - /usr/src/mm/table.c

Todo: add an entry into call_vec table to handle new system call. If you used an unused system call number, change entry (no_sys) to the function call implemented in misc,c and declared in proto.h. Otherwise, add to end as outlined in syscall handout.

Go to the end of the `call_vec array`

Add the line `no_sys, /* 78 = FOO */ at the end of call_vec`

Save and exit

Our new `call_vec should look like this:`

```
PUBLIC _PROTOTYPE( int (*call_vec[]), (void) ) = {
          no_sys,   /* 0 = unused       */
          do_exit,  /* 1 = exit                   */

          /* ... omitted code ... */

          do_svrctl,          /* 77 = SVRCTL      */
          do_foo,   /* 78 = FOO                   */
};
```

Example adapted from: http://wwwcsif.cs.ucdavis.edu/~cs150/syscall.html, written by Sophie Engle.

Changes to MM - /usr/src/mm/proto.h

Todo: add a declaration for the MM-side do_*yoursyscall*() function which will be implemented in misc.c.

Now, we should give MM the prototype for our new system call. This is done as follows:

- Open `/usr/src/mm/proto.h`
- Go to the `/* misc.c */` section (on line 41)
- Add the line `_PROTOTYPE( int do_foo, (void) );`
- Save changes and exit

The misc.c section should look like this:

```
/* misc.c */
_PROTOTYPE( int do_reboot, (void)       );
_PROTOTYPE( int do_svrctl, (void)       );
_PROTOTYPE( int do_foo, (void)                    );
```

Note: while your system call takes two parameters, this function call takes void. This is because the call_vec table in table.c takes no parameters.

Example adapted from: http://wwwcsif.cs.ucdavis.edu/~cs150/syscall.html, written by Sophie Engle.

Changes to MM - /usr/src/mm/misc.c

Todo: the misc.c contains the do_*yoursyscall*() function, which is responsible for passing a message into the system component in the I/O layer. This occurs in the sys_*yoursyscall*() function you added to the /usr/src/lib/syslib directory. Example using foo:

```
PUBLIC int do_foo()
{
          int num, num2;

          num = mm_in.m4_l1;
          num2 = mm_in.m4_l2;

          if( num > 0 ) {
                    printf( "syscall do_fork( %d ) called!\n", num );
                    return 0;
          }
          else {
                    return -1;
          }
}
```

In addition to retrieving the parameters from the message, you will need to call your sys_yoursyscall() passing those parameters and return the result to the user process.

Example adapted from: http://wwwcsif.cs.ucdavis.ed/~cs150/syscall.html, written by Sophie Engle.

Changes to Kernel - /usr/src/kernel/system.c

This file allows an interface to access the proc table within the kernel.

To do:

• add function prototype for system call

```
#include "kernel.h"
#include <signal.h>
#include <unistd.h>
#include <sys/sigcontext.h>
#include <sys/ptrace.h>
#include <minix/boot.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include "proc.h"
#if (CHIP == INTEL)
#include "protect.h"
#endif

/* PSW masks. */
#define IF_MASK 0x00000200
#define IOPL_MASK 0x003000

PRIVATE message m;

FORWARD _PROTOTYPE( int do_abort, (message *m_ptr) );
… code omitted ...
FORWARD _PROTOTYPE( int do_getmap, (message *m_ptr) );
```

Changes to Kernel - /usr/src/kernel/system.c

This file allows an interface to access the proc table within the kernel.

To do:

• add case in sys_task() to handle incoming messages. The values for the case statement are defined in /usr/include/minix/com.h.

•Add your kernel side do_yoursystemcall() at end of file.

```
PUBLIC void sys_task()
{
/* Main entry point of sys_task.  Get the message and dispatch on type. */
  register int r;

  while (TRUE) {
          receive(ANY, &m);
          switch (m.m_type) { /* which system call */
              case SYS_FORK:  r = do_fork(&m);    break;
… code omitted ...
              case SYS_TRACE: r = do_trace(&m);   break;
              default:                  r = E_BAD_FCN;
          }
          m.m_type = r;                 /* 'r' reports status of call */
          send(m.m_source, &m);         /* send reply to caller */
  }
}
```

Changes to Kernel - /usr/src/kernel/system.c

This file allows an interface to access the proc table within the kernel.

To do:

•edit do_fork and do_xit to handle process creation and deletion.

```
PRIVATE int do_fork(m_ptr)
register message *m_ptr;      /* pointer to request message */
{
/* Handle sys_fork().  m_ptr->PROC1 has forked.  The child is m_ptr->PROC2. */
… code omitted …
  register struct proc *rpc;
  struct proc *rpp;
  *rpc = *rpp;                            /* copy 'proc' struct */

… code omitted ...
  rpc->p_pendcount = 0;
  rpc->p_pid = m_ptr->PID;    /* install child's pid */
  rpc->p_reg.retreg = 0;      /* child sees pid = 0 to know it is child */

  rpc->user_time = 0;                     /* set all the accounting times to 0 */
  rpc->sys_time = 0;
  rpc->child_utime = 0;
  rpc->child_stime = 0;
… code omitted …
}
```

Changes to Kernel - /usr/src/kernel/proc.c(h)

To do:

- add variable to store number of lottery tickets to proc.h
- adapt pick_proc() function to implement lottery scheduling

```
… code omitted ...
PRIVATE void pick_proc()
{register struct proc *rp;    /* process to run */

  if ( (rp = rdy_head[TASK_Q]) != NIL_PROC) {
          proc_ptr = rp;
          return;
  }
  if ( (rp = rdy_head[SERVER_Q]) != NIL_PROC) {
          proc_ptr = rp;
          return;
  }
  if ( (rp = rdy_head[USER_Q]) != NIL_PROC) {
          proc_ptr = rp;
          bill_ptr = rp;
          return;
  }
  /* No one is ready.  Run the idle task.  The idle task might be made an
   * always-ready user task to avoid this special case.
   */
  bill_ptr = proc_ptr = proc_addr(IDLE);
}
```

Compiling Kernel and Libraries

To compile the kernel, do the following commands in Minix (will take awhile):

- `cd /usr/src/tools/`
- `make clean`
- `make`
- `make hdboot`
- `sync`

To compile the libraries, do the following commands in Minix (will take a long time):

- `cd /usr/src/lib/`
- `make clean`
- `make all`
- `make install`
- `sync`

The libraries only need to be compiled after /usr/src/lib/syslib/sys_setlot.c is added. Also, make clean does not need to be run each time you compile as this will cause the entire kernel or libraries to be recompiled. Omitting this step will only compile files you modified.

Taken from: http://wwwcsif.cs.ucdavis.ed/~cs150/syscall.html, written by Sophie Engle.

Compiling Kernel and Libraries

Since we are only adding a single file to the library, it is faster to compile that library and install that library file by itself.

Alternative to compiling entire library:

```
cd /usr/src/lib/syslib

make

cd ../

install -c -o bin libsys.a /usr/lib/i386/libsys.a
```

Compiling Kernel and Libraries

Hint: create a script to copy your files to the correct location and compile them. Then save in floppy. If your kernel crashes, you can run the script to restore. Example (chmod +x filename to make it executable):

```
#!/bin/sh -e
cp Makefile sys_setlot.c /usr/src/lib/syslib/
cp misc.c proto.h /usr/src/mm
cp table.c.mm /usr/src/mm/table.c
cp table.c.fs /usr/src/fs/table.c
cp proc.h proc.c system.c /usr/src/kernel
cp unistd.h setlottery.h /usr/include
cp callnr.h com.h syslib.h /usr/include/minix
cd /usr/src/lib/syslib
make
cd ..
install -c -o bin libsys.a /usr/lib/i386/libsys.a
cd /usr/src/tools
make
make hdboot
sync
```