<u>README Programming Assignment 2 Winter 2015</u>
Author: Dinesh Jayasankar
<u>Problem #1</u> Demonstrate with F7 key.

/usr/src/servers/is/proto.h
In proto.h, I declared _PROTOTYPE(void magic_dmp, (void));

/usr/src/servers/is/dmp.c
I assigned magic_dmp to function key F7 in struct hook_entry hooks[NHOOKS]

/usr/src/servers/is/dmp_kernel.c
I defined PUBLIC in messages[NR_TASKS + NR_PROCS][NR_TASKS + NR_PROCS];
I define PUBLIC void magic_dmp(). Inside magic_dmp(), I retrieved messages matrix with sys_getmagic(messages). The outer for-loop prints the first all the processes 10 at a time. In the inner for-loop, for each process printed, I print the first 10 destination processes that receive a message. If the source or destination process is shutdown, which is checked with isemptyp(), I skip that iteration of the loop.

/usr/src/kernel/proc.h
I declared the matrix EXTERN messages[NR_TASKS + NR_PROCS][NR_TASKS + NR_PROCS].

/usr/src/include/minix/syslib.h
I declare sys_getmagic(dst) as a shorthand for sys_getinfo(GET_MAGIC, dst,0,0,0)
It gets a copy of system info or kernel data.

/lib/syslib/sys_getinfo.c
This contains PUBLIC do_getinfo(...)

/usr/src/kernel/system/do_getinfo.c
This contains PUBLIC int do_getinfo(m_ptr).
I have a case statement "case GET_MAGIC". The length is initialized to sizeof(int) * (NR_PROCS+NR_TASKS) * (NR_PROCS+NR_TASKS). I use vir2phys() to get the physical bytes of the integer message matric.

/usr/src/kernel/proc.c
I increment the appropriate entry in the two-dimensional message matrix inside mini_send(). The row index is caller_ptr->p_nr + NR_TASKS, and column index is dst_ptr->p_nr + NR_TASKS

include/minix/com.h
NR_TASKS is defined here and IDLE is -4. Since Kernel tasks have negative process numbers, we require the NR_TASKS offset when indexing the message matrix.

<u>Problem #3 </u>Demonstrate with F11 key
/include/minix/callnr.h (page 676).
#define NRCALLS 91 is the number os system calls allowed. The user program's typical system calls range from 1 to 63.
/usr/src/kernel/proc.h
I declared the fields "int calls[NCALLS]" to record each instance of a system call and "int num_of_calls" to record system call frequency.

/usr/src/other/syscall.c
I modified _sendrec(...) to accept three parameters: who, msgptr, syscallnr
The function _syscall handles all system calls from user programs.
_syscall takes as parameters who, syscallnr, and msgptr.

Example:
- ❖ if (fork() != 0)
- ❖ */lib/syscall/fork.s*
  - ➢ ··· . _fork: jmp _fork defined in *src/lib/posix/fork.c*
- ❖ */src/lib/posix/fork.c*
  - ➢ ... return(_syscall(PM, FORK, &m));

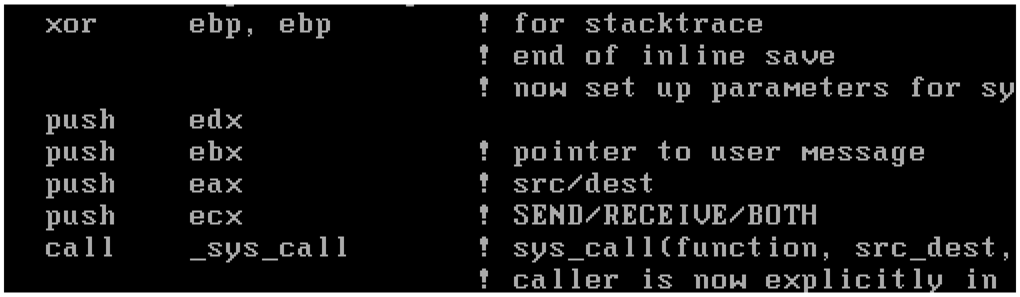
_syscall calls _sendrec(who, msgptr).
NOTE: _sendrec calls the assembly code in lib/i386/rts/_ipc.s or lib/i86/rts/_sendrec.s

/usr/src/include/minix/ipc.h and /include/minix/ipc.h
I changed _sendrec's prototype to _PROTOTYPE(int sendrec, (int src_dst, message *m_ptr, ···));
Note the default arguments. This is possible due to #include <stdarg.h> (variadic functions).

/lib/i386/rts/_ipc.s and /lib/i86/rts/_sendrec.s
NOTE: Documentation says that edx is unused.
_ipc.s:
I added line "mov edx, CALL_NR(ebp)" to store the extra syscallnr parameter.. CALL_NR is 16, because the parameter syscallnr comes after msgptr.
_sendrec.s
I added the line "mov dx, 8(bp)" to store the extra syscallnr parameter. Note that this is not using extended register notation, so the offset different will be by 2, instead of 4.

/kernel/mpx386.s (_s_call)
NOTE: The register edi is unused, but the parameter is in edx on the stack.
Change all references of dx to di.
mov di, ss
mov ds, di
mov es, di

Place edx in the right place using “push edx”. Now, sys_call can access this as a normal parameter.



```
xor     ebp, ebp        ! for stacktrace
                        ! end of inline save
                        ! now set up parameters for sy
push    edx
push    ebx             ! pointer to user message
push    eax             ! src/dest
push    ecx             ! SEND/RECEIVE/BOTH
call    _sys_call       ! sys_call(function, src_dest,
                        ! caller is now explicitly in
```

/kernel/proto.h
Modify the parameters of syscall to include call_index which will hold the user-space value of syscallnr.

/kernel/proc.c
If the system call number is between 1 and 63, the function is a SENDREC, and proc_ptr is a user process then use call_index (syscallnr) to index the field “calls” and increment it.

/kernel/system/do_fork.c
On a new process fork reset the calls and num_of_calls field!

Problem #2 Demonstrate with F1 key.
/kernel/proc.c (sched)
If it is a user process and its system and user time are 0, then set “time_left” to 1 and give the new process a new quantum. A fork inherits the parent's existing p_ticks_left.

This sets *front = time_left -> 1.

/kernel/proc.c enqueue
Since front is 1, the new process will be placed at the front of the queue.