Advanced Operating System. Assignment

Q1: Why we have to use another kernel source, can we compile the original kernel?

Answer:

I can't compile the local operating system because it's alrdeadly complied. The local operating system is in a binary format and couldn't be compiled. I need to download another copy, source format, then I need to build the binary of this source code in a given directory on my disk.

Q2:

What is meaning of other parts, ie. i386, procsched, and sys_procsched?

i386: ABI (Application Binary Interface)

procsched: name of the syscall.

sys_procsched: entry point.

Q3:

Open the file include/linux/syscalls.h and add the following line to the end of this file:

```
struct proc_segs;

asmlinkage long sys_procsched(int pid, struct proc_segs * info);
```

QUESTION: What is the meaning of each line above?

The 1st line:

Struct proc_segs; // proc_segs arguments of type Struct.

The 2nd line:

long : function handling syscall return type long.

Asmlinkage tag is a #define with some gcc magic tells the compiler that the function is not expected to optimally find all the arguments in the registers.(a common optimization), but only on the CPU's stack.

System_call consumes its first argument, the system call number, and allows up to 4 more arguments passed to real system, on the stack. All system calls are marked with asmlinkage tag, so they all look to the stack for arguments.

It's also used to allow calling a function from assembly files.

Q4.

First run "make" to compile the kernel and create vmlinuz. It takes a long time to "$ make", we can run this stage in parallel by using tag "-j np", where np is the number of processes you run this command.

```
$ make
or
$ make -j 4
```

vmlinuz is "the kernel". Specifically, it is the kernel image that will be uncompressed and loaded into memory by GRUB or whatever other boot loader you use.
Then build the loadable kernel modules. Similarly, you can run this command in parallel.

```
$ make modules
or
$ make -j 4 modules
```

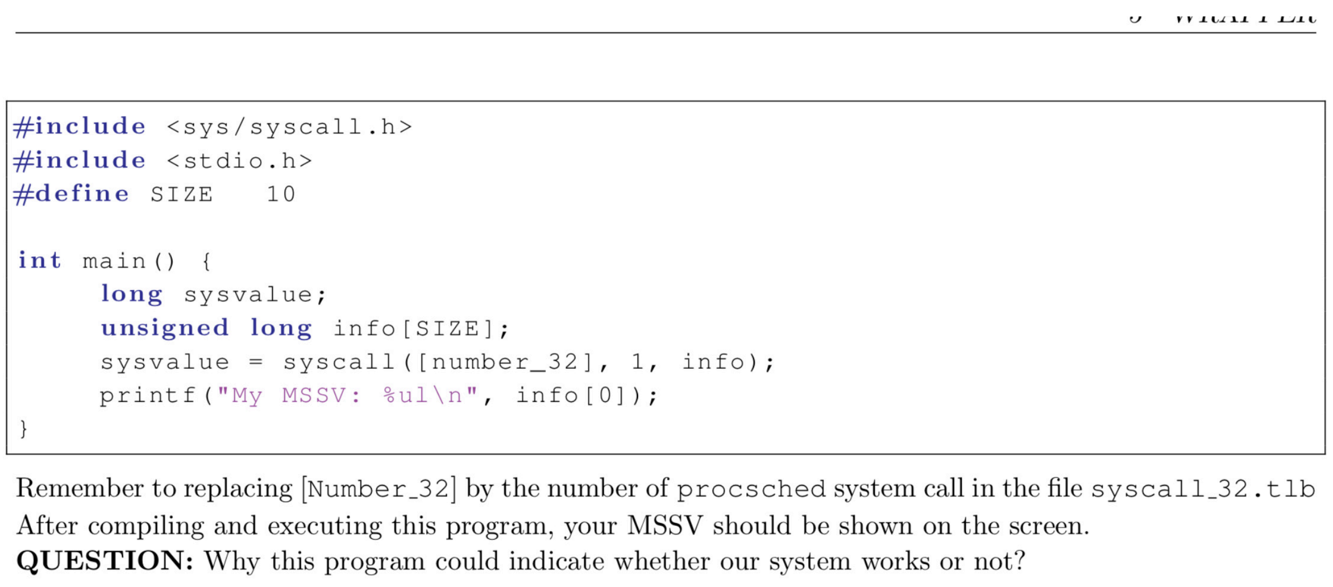
QUESTION: What is the meaning of these two stages, namely "make" and "make modules"?

"make" : compile and link the kernel image. There is a single file name "vmlinuz"

"make modules": comiples individual files for each question you answered "M" during kernel config. The object code is linked against your freshly built kernel. (For question answered "Y," these are already part of "vmlinuz". and for questions answered "N" are skipped).

[illegible]

Q5.



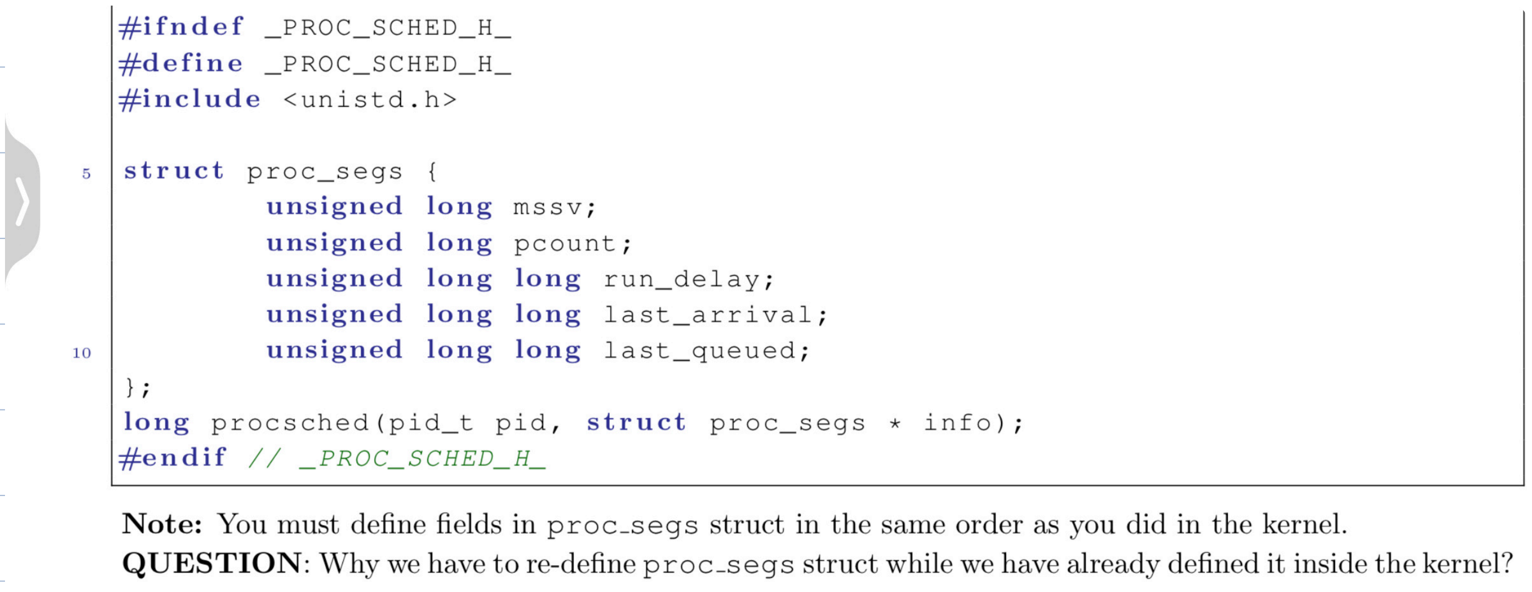
```c
#include <sys/syscall.h>
#include <stdio.h>
#define SIZE   10

int main() {
    long sysvalue;
    unsigned long info[SIZE];
    sysvalue = syscall([number_32], 1, info);
    printf("My MSSV: %ul\n", info[0]);
}
```

Remember to replacing [Number_32] by the number of procsched system call in the file syscall_32.tlb
After compiling and executing this program, your MSSV should be shown on the screen.
QUESTION: Why this program could indicate whether our system works or not?

This program is to check if system call has been integrated into kernel. by calling "syscall ([number_32], 1, info)", "info[0]" should be the "MSSV" because "mssv" is defined in the first line of the "struct proc_segs".

Q6.



```c
#ifndef _PROC_SCHED_H_
#define _PROC_SCHED_H_
#include <unistd.h>

struct proc_segs {
        unsigned long mssv;
        unsigned long pcount;
        unsigned long long run_delay;
        unsigned long long last_arrival;
        unsigned long long last_queued;
};
long procsched(pid_t pid, struct proc_segs * info);
#endif // _PROC_SCHED_H_
```

Note: You must define fields in proc_segs struct in the same order as you did in the kernel.
QUESTION: Why we have to re-define proc_segs struct while we have already defined it inside the kernel?

Because we have leave out kernel source code directory and create anothere directory to store the source code for our wrapper. The re-define of proc_segs is for the prototype of the wrapper. They are in different directory with the one defined inside the kernel.

Q7 **Validation** You could check your work by write an additional test module to call this functions but do not include the test part to your source file (`procsched.c`). After making sure that the wrapper work properly, we then install it to our virtual machine. First, we must ensure everyone could access this function by making the header file visible to GCC. Run following command to copy our header file to header directory of our system:

```
$ sudo cp <path to procsched.h> /usr/include
```

QUESTION: Why root privilege (e.g. adding sudo before the cp command) is required to copy the header file to `/usr/include`?

The command "cp" needs to be run with a root privilege.
We are copying our header file "procsched.h" to header directory of our system so that everyone could access this function.
This changed the privilege of the file, therefore, needs the root privilege "sudo"

Q8.

```
$ gcc -shared -fpic procsched.c -o libprocsched.so
```

If the compilation ends successfully, copy the output file to `/usr/lib`. (Remember to add `sudo` before `cp` command).
QUESTION: Why we must put `-share` and `-fpic` option into `gcc` command?.

-Share: indicates that a shared library is generated;
- fpic: indicates that using position independent code.

The shared object can be loaded to different positions by different proceses.

The purpose is to fullfill the request that compile source code as a shared object to allow user to integrate our system call to their applications.