ASSIGNMENT 4A

1. What are the ELF magic numbers?

Answer: The following sets of files are the ELF number:

elf mag 0-3 , ELF OSABI, ELFOSABI\_SYSV, ELFOSABI\_STANDALONE, Elf PAD

1. What is the difference between UIO\_USERISPACE and UIO\_USERSPACE? When should one use UIO\_SYSSPACE instead?

ANSWER-

UIO\_USERISPACE - source destination for USER PROCESS CODE. The user Level segments of memory are being referenced are executable.

UIO\_USERSPACE- source destination for user process data

UIO\_SYSSPACE- source destination for kernel i.e when the memory being referenced are inside the kernel.

We will use SYSSPACE when the kernel space is to be set as NULL.

1. Why can the struct uio that is used to read in a segment be allocated on the stack in load\_segment() (i.e., where does the memory read actually go)?

Ans- The data related to the file transfer is defined. E.g. the block of data which is required by the IO , the number of available data blocks which are involved the remaining amount of data to be transferred from user to kernel space or from kernel space to user space. While reading the memory, the offset would go to read which UIO has specified and UIO\_RESID is getting updated and according UIO\_OFFSE is matched.

1. In runprogram(), why is it important to call vfs\_close() before going to usermode?

Answer: Because a file opened in kernel space cannot be opened in user space too.

1. What function forces the processor to switch into usermode? Is this function machine dependent?

Ansqer: enter\_new\_process is the method.

1. In what file are copyin and copyout defined? memmove? Why can't copyin and copyout be implemented as simply as memmove?

Answer: copyinout defines the copyin and copyout functions.

Memmove defined in /common/libc/string/memmove.

Copyion and copyout moves the data back and forth between user and the kernel space.

1. What (briefly) is the purpose of userptr\_t?

Answer: userptr\_t is used to check whether the block of user memory falls within the proper user space fregion while its copying data back and forth between the kernel space.

1. What is the numerical value of the exception code for a MIPS system call?

Answer: the numerical value is 8.

1. How many bytes is an instruction in MIPS? (Answer this by reading syscall() carefully, not by looking somewhere else.)

Answer: 4 bytes of instructions.

1. Why do you "probably want to change" the implementation of kill\_curthread()?

Answer: because the one which is currently implemented is not handling the actually killing of the thread.

1. What would be required to implement a system call that took more than 4 arguments?

Answer: Passing a pointer to a block of stack that contains all the args after the first two or 4 args depending on the size of each arguments

1. What is the purpose of the SYSCALL macro?

Answer: syscall value loaded into v0 register and then it goes to syscall code in MIPS.

1. What is the MIPS instruction that actually triggers a system call? (Answer this by reading the source in this directory, not looking somewhere else.)

Answer:syscall is the instruction that triggers it.

1. After reading syscalls-mips.S and syscall.c, you should be prepared to answer the following question: Now that OS/161 supports 64-bit values, lseek() takes and returns a 64-bit offset value. Thus, lseek() takes a 32-bit file handle (arg0), a 64-bit offset (arg1), a 32-bit whence (arg3), and needs to return a 64-bit offset value. In void syscall(struct trapframe \*tf) where will you find each of the three arguments (in which registers) and how will you return the 64-bit offset?

Answer: 32 bit file will handle the a0 register. 64 bit offset in a2 and a3. 32 bit when the values are needed to be copied from copyin function e]which uses the trap frame pointers.

SYSCALLS----

fork()

-Creation of a new address space, trap frames

-creating a new pointer thread and for memory allocation.

-allocating size for the trap frame and then during the syncronization process the piud gets updated because of the increments of the ciunter.

-the pid of parent equals to the pid of the newly created thread pid.

-callinmg the thread fork function to store the result and the returning the pid.

getpid()

* We get the pid of the current process from yhe current thread.
* And then the value is returned.

waitpid()

1. Checking if the status pointer is aligned and if its valid

2.check if the options are valid.

3.checking for the childpid whether its valid or does it exists.

4.check if it that childpid belongs to our child

check the status pointer for kernel(Error Handling stuff)

5.Acquire the lock and then check on the child, if not exited the

continue waiting on CV.

Then if the child has exited the destroy it.

Or

Wait till the child exits.

Once done then destroy it

6. Release the lock

exit()

Process checks if its children are in ZOOMBIE state.

If yes then it kills it

If no the the are orphaned and the process itself turns to ZOOMBIE.