PROJECT 2 FREQUENTLY ASKED QUESTIONS

1. **How do I change the Makefile ?**

For project 2, you need to know how Makefiles work. Look at the Makefile in the code/test/ directory to see how halt.c is compiled and the coff and noff files produced. Your test programs for testing the new system calls should be compiled in a similar way. You can add to the existing Makefile the rules for compiling the new test programs. Here is tutorial on Makefiles. Just read the basics. Look at the example presented:
http://theory.uwinnipeg.ca/localfiles/infofiles/make/make_toc.html

2. **How do I use the gdb debugger ?**

You should look at gdb documentation to figure out how to use the debugger to its full extent. For a start, you can use the following commands:

At the prompt type: **`gdb ./nachos`**
Once inside the debugger, the following command will run the Halt program: **`r -x ../test/halt`**

You can substitute the part after '`r`' for your own command line

3. **I have included "somefile.h" in myFile.cc but it still says it's not defined! What do I do?**

This is a problem of circular includes. Please look at the compile error message to figure out what is the circle of .h files which causes the error. Then in the actual file which causes the error condition (for most people this would be system.h) add class declarations before you include any .h files. These would be:

class Thread;

If you need any more declarations, add them based on the error messages.

4. **What should I start implementing first?**

I suggest starting by creating a class/structure for memory manager, a process control block, and a process control block table. I would then start by implementing the system calls in the following order: Yield, Exit, Join, Exec, and Fork. Yield is very simple so shouldn't take you any time at all. Exit and Join are related so it helps to do them together. Fork is the most difficult (along with Exec) so it helps to do them after the others.

5. **How can I test (why doesn't printf work in my test programs)?**

You will have to test based on flow of execution. For example: to test if Fork/Exec works you Fork/Exec a function/executable and then call Halt() in the function/executable. If Nachos halts when you run your test program then your system call is probably working. Use DEBUG statements for printing information to the screen.

6. **What does the memory manager do?**

It is merely a way of assigning pages from the system's (Nachos) main memory. It does not actually allocate memory (via malloc/new or some equivalent). Basically, you will want to enforce atomic (synchronized) operations on a bitmap that represents the number of pages in main memory (there are 32 pages in Nachos main memory right now).

7. **What does a process control block contain?**

A process control block (pcb) contains the attributes of a process. Some of the major attributes are the thread, the pid (SpaceID), open files, etc (remember that the thread has a pointer to the addrspace which is an indirect attribute of a pcb). There should be a global table of process control blocks as well. Remember that you will also want to be able to get a particular process's condition so that it can be waited on in Join (if necessary) and broadcast in Exit.

8. **How do I add a new file for this project?**

You will want to create your .h and .cc file in userprog (every new file for this project can be added in this directory). Then you will want to add the .h, .cc, and .o to the list of files in the top level Makefile.common in the correct USERPROG section. Be sure not to add your file to the end of the list of files, but instead place it in the same section as the other userprog files. In each USERPROG section (ie _H, _C, and _O) all the files from each directory should be placed with the other files from that directory. Once this is done type "make depend" in the userprog directory and then you will be able to type "make" to compile as before.

9. **How do I translate the name of the executable when implementing Exec?**

You will want to put a translate function in addrspace that is similar to the translate

function in Machine. You will use this function to translate a virtual address into a physical address (one page at a time).

10. Is there a difference between the parameters of Exec and Fork?

Yes, there is. Exec takes the string representing the relative path (from where you run Nachos - userprog in this case) to the test program you wish to exec [ie. Exec ("../test/myExecedProgram")]. Fork takes the name of the function you wish to fork. It is not a string, but a function pointer so it has the form Fork(myFunction) where you have implemented void myFunction() previously in you test program. In Exec you are translating the string that is passed in and in Fork you are using the pointer to myFunction as the PCReg value for when you run the new thread. If you try to pass a string to Fork or a name (not in string form) to Exec you will have serious problems

11. Does a Forked thread use the same addrspace as the thread who Forked it?

No, it does not. It uses a COPY of the addrspace from the thread who Forked it. This means you must create a new addrspace that has the same size page table as the Forking thread and then you must copy the Forking thread's pages in memory into the Forked threads pages in memory.

12. The project spec says we need to add a function ReadFile to Addrspace. What is this used for?

This is used for copying the executable's code and data segments into memory (in the constructor for Addrspace (that Exec calls). noffH.code.virtualAddr is the logical address of the executable's code segment and noffH.initData.virtualAddr is the logical address of the executable's data segment. You no longer want to zero out (bzero) the memory. You need to copy these sections page by page into main memory. You will use *your* Addrspace::Translate to translate the logical data/code address to the physical address (in memory). Then you can use the C/C++/Unix "bcopy" to copy it over.

13. Where should we put the global structures/classes?

You can put these in system.h in the threads directory. Only put the global variables (their declaration) in this file. Make sure you actually instantiate the global structures (like memory manager) in the system.cc file in the initialize function under the correct #ifdef's (for userprog in this case).

14. Does the Exit system call take a parameter? What is that parameter?

The Exit system call should take an integer parameter. This parameter is the exit value of the process. This is the same as the exit values in Unix (0 -> good, 1 -> bad). For our purposes you can use any integer here. It only matters that if another process was "Join"ing on your process that the Join call would return the value that your process exited with (as per the

project spec). This means that you need to save your exit value somehow in the Exit system call so that Join can return it if necessary (even if you've already exited when someone calls Join on your process).

15. **What should we do when we can't allocate enough physical pages in Addrspace constructor for a new thread?**

You should let Fork/Exec know that you don't have enough memory so that it can let the user know (return pid of -1) that it didn't Fork/Exec a new process. Make sure that you check if there are enough free pages in physical memory to facilitate the number of pages for your new process before trying to allocate the physical pages for each logical page. If you do not do this one of your allocates will return -1 and you will have to deallocate all the pages you just allocated before you can let Fork/Exec know of the error.

16. **Do Exec and Fork call thread->Fork?**

Yes, they do. After you have set all other information up (as in the project spec) you need to thread->fork a dummy function (that's in exception.cc). In this dummy function you will want to initialize and restore the registers (for Fork and Exec) and then set up the PC registers and return register address (for Exec only). After this you will call machine->Run() from the dummy function (for both Fork and Exec). Exec may not create a new thread if you choose to implement it by replacing current state of the thread instead of deleting and creating a new one with the same pid.

17. **Should I implement any TLB functionality?**

No. You don’t have to add any TLB functionality. By default, the TLB should be disabled for Nachos. All the changes for Nachos must be made from the point of view of the operating system and not from the hardware perspective.

18. **Why use bitmap?**

Nachos has an entire class implemented called bitmap inside …/code/userprog directory. This class allows you to access each element of an array at fine granularity (1 bit). For this project, you will use bitmap to keep track of the page allocations on main memory. Read all the comments in bitmap.h and bitmap.cc files to learn more about it.