Before you start working on project 3 ....

1. Watch Jason's video on the inners of NachOS

https://youtu.be/F2TLCjwe1kQ

2. Watch Jason's video on system calls and exceptions in NachOS

https://www.youtube.com/watch?v=rXBz7a0HiyQ

3. Watch this short video on tips for group work

5 Expert Tips for Group Success - YouTube

Design Summary

Q. Are we going to be graded if we get the design specs wrong or if we are going in the wrong direction. Or is this all just feedback to help us

Answer: The design summary's primary goal is to give students time to study the provided specs, go through the code, and make a plan as a group on how to proceed. Just give an idea of your understanding of the concept and write some pseudocode if possible for each task. Once you have submitted the design summary, we would provide feedback on whether you are on the right track or not. We will also make some more detailed posts on each task. So, do not delay the design summary. The key is to organize as a group and dividing work.

Handling programs that are too big to fit.

The physical memory size is 32 frames. A program can be bigger 32 frames or if multiple programs are running as in multiple join or shell programs, the incoming program may be larger to not fit by any type of fit options. You can yield the incoming program thread and try couple of times to see if space is available or you can simply let the thread die. Just prompt on the screen about the information.

NOFFMAGIC Question

Q. Upon trying to execute a user program there is an assertion call that fails within addrspace.cc

I see that the NOFFMAGIC is hardcoded to be 0xbadfad

When running the user program halt.c,

noffH.noffMagic = 1746938415

WordToHost(noffH.noffMagic) also equals 1746938415

It seems that within WordToHost the initial parameter passed is just being returned instead of any bitshifting taking place, which leads to noffH.noffMagic never equaling NOFFMAGIC, leading to an assertion failure.

ASSERT(noffH.noffMagic == NOFFMAGIC)

In the comments the author writes that NOFFMAGIC is the magic number which denotes a Nachos object code file.

So it seems that the failed assertion implies that the userprogram is not being detected as a Nachos object code file.

Answer: You are supposed to run the compile file and not the code file i.e. .c file.

You need to compile halt.c file and then run the compiled file. If you create a new program in test you must add instructions to the MakeFile before compiling. See previous instructions for matmult, sort, etc already in the MakeFile.

Contiguous memory

Q. Does all of a single process's memory need to be contiguous for Task 3 onward? As it stands right now, our code is working, but a process isn't being saved contiguously in memory.

No, Just focus on loading the Page for which we get page fault. We are just following the concept of page fault that you learned in the class lecture.

Task 2

Q. What is the purpose of task 2?

Task 2 is related to the Addrspace constructor. All the instructions in task 2 were to be done at the start of the project to understand the process and have a complete section by itself. However, after integrating all parts of the project, there are only a few things that you do here. Basically, the instructions that you follow for task 2 get divided into other sections.

My post on the project summary gives the final product outline. As a final product, the things that you do here come down to the following.

- After the size is calculated, you just need to do the following things
    - create a swap file based on id and copy the content of the executable into it.
    - No need to find page offset here or copy the content of executable to main memory. We handle that during page fault exception.
    - In the pagetable, do not set any value on the physical page (we assign that during page fault)
    - Necessarily, set the valid bit to false (it causes page fault and there you can load a page at a time)

Just to test task 2 and understand copy of content into main memory, you could have used a continuous memory allocation scheme. However, since we are using page fault mechanism based on -V option, we are either ending the program or replacing a page of main memory used by another thread or other pages of the same thread, the idea of memory allocation is no longer required.

Q. My group had some confusion on Task 2 around which variable to use for the pageTable array. Should we be using i or freePage on the left side? for example should it be pageTable[i].virualPage or should it be pageTable[freePage].virtualPage? Also, how do we go about differentiating the virtual pages and physical pages?

There is already an implementation of pagetable in the base code in addrspace constructor. See how it is implemented. Also, see my post on project 3 details and read through it properly.

If you see in addrspace.cc, under addrspace constructor, there is an implementation of pagetable as below

```
// first, set up the translation
pageTable = new TranslationEntry[numPages];
for (i = 0; i < numPages; i++) {
      pageTable[i].virtualPage = i;
     // pageTable[i].physicalPage = I;  // We no longer need this line or it is irrelevant as we set valid
bit to false
      pageTable[i].valid = FALSE;        // We set this valid bit to false as we are not loading any content
into memory
      pageTable[i].use = FALSE;         // we will handle loading page during page fault. Leave other
options as it is
      pageTable[i].dirty = FALSE;
      pageTable[i].readOnly = FALSE;

}
```

Q. I am not sure where startprocess() is being called and I am wondering what is going on when there are multiple processes. As in, I see that startprocess() takes in one file name. So startprocess() take a parameter of an array of filenames or where ever startprocess() is being called is there a loop where a filename would be input one by one if you want mutiple processes?

StartProcess is first called from main.cc, in the part where it tests if you used the -x flag. This will start up the first user processs (the one indicated in the command line), which can then start up other processes. You won't want to pass an array of file names since you won't know what all user programs will be executed when you first start. Imagine our first user program execs 3 programs, each of which exec 2 other programs. There's no way to know all of that from the start.

Notice however that the code from the StartProcess function is replicated in the SC_Exec section in exception.cc. This code will be called when a user program triggers the Exec exception, and takes the passed parameter to Exec(), which is the program the first user program is trying to execute, and runs it through the same process StartProcess does.

Task 2 Deallocation

Q. Where are we supposed to deallocate memory, so that it lets the program finish running before it deallocates it?

Answer: deallocation means just clearing the bits used by the exiting thread. You only have to do is find the bits used by the exiting thread and then clear that bit. You can refer to IPT to find the bits used by the exiting thread and just clear them.

Task 3 accessing executable for ReadAt()

Q. I'm having difficulties wrapping my head around how to access the Openfile executable from within PageFaultException to do a readAt as I cant seem to figure out where the executable is stored or if it's stored at all. I thought it may be stored somewhere within the current thread but had no such luck tracking it down. If it is not stored, how should I go about recreating it?

executable is just a file pointer to the program file. The default program files are in the test folder.

./nachos -x ../test/sort

If you run the program as above, "../test/sort" is the program that you want to run. In this case, your main program is sort and if you see the code you will see it just computes the sort algorithm and exits by passing the value as an argument. Therefore, your program executes as below.

starts in main, because of -x goes to progtest.cc, there you call addrspace for first time. Since we always set the valid bit of pagetable to false, we will always get pagefault and the number of pagefault will be equal to the number of pages that the sort program needs. And in each page fault we will copy one page from the program file (executable or swap file) into the main memory.

After the successful copy of a page,

- update pagetable corresponding valid bit to true.

- update fifo list
- update IPT table
- If you are replacing page make corresponding updates too

Now, if you are running program like shell.

./nachos -x ../test/shell

Here main program is shell, it requires around 6 pages and so you will get 6 page faults. When the program runs you will be prompt to enter file name.

If you enter the file name correctly it will execute Exec and run that program. You will need to enter ../test/sort to run sort program on the shell program.

Q. I am still having a hard time understanding how to implement this part. From what I understand, I want code similar to the "executable->ReadAt..." (found in addrspace.cc) and incorporate it in my PageFaultException Case. I can't seem to understand how to send the executable pointer from addrspace.cc to exception.cc so I can complete the next step which is *"Finally, if space was found, load the single page into mainMemory using ReadAt()"*.

To handle this it may be useful to think back to your object-oriented roots.

Recall that each thread is an object which has a "space" member of AddrSpace type. There's not some global address space for all programs, each one has its own, with members like a pageTable and member functions like InitRegisters.

So following that, you could have a member in AddrSpace for the executable object so that address spaces could just access it all the time. (Keep in mind though that you'll have to transition to swap files when you want to do tasks 4/5.)

You could also have a member function for AddrSpace that will handle page swapping, maybe you give it the parameter for a page to put in, or you could just access the executable object directly from exception.cc, the choice is yours.

All of this would then be accessible using currentThread->space->executable/PageSwap()

This is not the only way to do things though, be sure to explore different methods to find what may be the best for you to understand and most efficient.

Task 5 - Initialization of List for FIFO Replacement

Q. I am having trouble figuring out where you would initialize the List for FIFO replacement?

Where would you initialize any variable? It depends on how you want to implement. For simplicity, we work with fifo list in exception.cc and therefore it is wise to define and use in exception.cc .

Exit()

Q. At the end of each test program, there is an Exit(int) function. Can anyone explain what should the int argument of this function?
In task 2, If I use the current implemented version of matmult which has
Exit(C[Dim-1][Dim-1]);
I will get abnormally exit error. But if I change it to Exit(0) it would work without any error message.

It is up to you what you want to pass the value. **It is just a notion to see argument 0 as a normal exit and anything else is an abnormal exit.** The base code does not print the argument value, it is best to print the value to see if you are getting the correct value. In matmult the default code should give something like 7220 as output which is normal and the program is running fine but in the code, we see that as an abnormal value as it is not 0 and so it prints as an abnormal exit.

ReadAt()

Q. I am confused with ReadAt implementation.

Assume that I want to copy the code segment of the executable file into the main memory. So, we have a line of code as follows:

executable->ReadAt(buffer,size_to_copy, offset_to_read_from_executable);

Here, the buffer should be a chunk of memory of size size_to_copy? If the buffer was a set of free physical pages but not contiguous, is it possible to copy the whole content to the buffer in one line? Or, we should do it in a loop, page by page.

executable->ReadAt(buffer,size_to_copy, offset_to_read_from_executable);

Here, it means start reading from the offset_to_read_from_executable and copy the content from the executable into the memory location specified (here buffer) and copy give size_to_copy.

—

Understanding ReadAt and WriteAt.

To copy content from a pointer executable to a buffer, we do like this

executable->ReadAt(buffer,size_to_copy, offset_to_read_from_executable);

The above statement will copy the content of the executable of the given size_to_copy into the buffer from the given offset.

Now, if I want to write the content from the buffer to executable, we do

executable->WriteAt(buffer, size_to_copy, offset_to_write_on_executable);

During page fault, you are copying a single page (bad virtual page) from executable or a swap file into the main memory (free physical page). So this is how we do it.

executable->ReadAt(&(machine->mainMemory[freePhysicalPage*PageSize]), PageSize, badVirtualPage*PageSize);

Here, ReadAt first argument should be the location where we want to write the page, which is given by

&(machine->mainMemory[freePhysicalPage*PageSize]) .

Since we are writing a single page, our second argument of size should be PageSize .

The third argument should be the offset from where to copy and which is given by
the badVirtualPage*PageSize

Now, if a page is dirty we need to write the content of the main memory into the swap file of the replacing thread. so we just do that using write at.

executable->WriteAt(&(machine->mainMemory[replacingPhysicalPage*PageSize]), PageSize, badVirtualPage_of_replacing_thread*PageSize);

—

Bitmap Initialization

Q. In memory allocation how big should we initialize the bitmap variable?

Answer: Bitmap is already initialized by default to NumPhysPages in Nachos. You do not need to change it. The Number of Physical page is limited to 32 in machine.h. You are not required to change it. Just check and mark bitmap bits as you need.

Q. If the size of our bitmap is relative to the number of programs running, how do we know what to initialize the size of the bitmap to? I understand the bitmap will be populated with each new program, but I'm just not sure how many items(page spaces) should be initialized to accommodate them all.

Answer: No, the bitmap is not different for different programs. The bitmap should be global and used by all threads. The bitmap should be initialized to NumPhysPages.

Bitmap represents the bits for NumPhysPages. Find() function of bitmap return the free bit and also sets the bit. There is no need to manually set the bit. However, you will need to clear the bit used by the exiting thread.

To define the global variable, add following lines.

In system.h, add

```
extern BitMap *bitMap;   //Declaring a global variable
```

In system.cc, add

```
BitMap *bitMap = new BitMap(NumPhysPages);  // defining the variable
```

After you have added the above two statements, you can use the object bitmap almost in any location you will have to make changes.

To find a bit just do,

int freePage = bitMap->Find(); // As I said above, it will return available bit from 0 to 31. If not available it returns -1.

To clear the bit just do.

bitMap->Clear(bit_number);

Q. When trying to define bit map in system.cc i can't use NumPhysPages. In what header is NumPhysPages declared?

Answer: machine.h

bzero.

Q. I'm working through task 2 and having trouble trying to figure out how this works in addrspace.cc. Where is bzero defined? I want to change its values but I'm not sure how it works.

Answer: There is no need to zero out any memory. For this project, we are handling all pages through page fault exception and so you will be adding one page at a time as required. Check my post on project 3 design summary on what are the essential steps to do for each task on different files.

Is the heap stored in a swap file?

Q. The description for Task 4 does not mention if the heap will be stored in the swap file which confuses me a bit. Is there a reason this would be excluded, or is it just not specified in the paper?

"The swap file must be large enough for all memory required by the user process. This includes code, initialized data, uninitialized data, and the stack."

Answer: The clean code has already code to calculate the size of the user process. You can simply use that to create a swap file. The original user program that is generated by nachos under the test folder puts additional content at the start of the user program that we use to verify if the program is compatible with nachos or not. If you see in the addrspace constructor we check NoffHeader of the file for it. The swap file can therefore simply be of a smaller size than the actual executable. See post on the design summary for more details.