**Task 2: System Calls**

**Exec()** - Exec will create a thread for a user program to run on. Exec will check to see if there is enough memory to run the program, and if so, it will get the filename and open the executable, and then allocate the memory and make a thread to run it. If not, an error message will be shown displaying proper output.

* Create a new address space for the process
* Check available memory
* If available memory < memory needed
  + Inform the user that there is not enough space. Then ASSERT & leave
* Else, create a thread, get file name using fileSystem->Open and allocate memory in the address space.
* Initialize registers and machine state for the thread.
* Use machine->Run() to run the process

**Join()** - Join will create a child thread to run a second program from the thread of a first program, putting the parent program to sleep. The ID of the parent, among other things, needs to be known, so that when the child is done running, it can wake up it’s parent. thread.h/.cc will need to be modified to include:

* How many children it has
* A parent pointer
* A bool indicating whether or not the parent is waiting to join with it
* What resources it is using
* A unique ID

A process control block will be implemented in progtest, which is just a list of active processes.

The actual joining will involve incrementing the number of children the current thread has, setting the thread with the given ID’s parent to the current thread, running the given thread’s ID, and putting the current thread to sleep.

**Exit()** - Exit will quit a user program. Resources need to be cleaned, such as deallocation of memory and thread finishing. The thread should check to see if it’s parent is asleep from a join, and if so, wake that thread before finishing. Exit should note if the program ends normally/abnormally depending on the status it is passed (0 being normal). We will not wait for its children to finish. Instead we will sever the parent/child relationship by setting their parent to null.

**Yield()** - Yield will essentially yield the thread that the user process is running on, allowing other threads to continue.

**Task 3: Address Space**

**StartProcess**

* Check executable file name
* Output if name doesn’t exist/NULL
* parent->space = child
* Put parent to sleep until child exits

**AddressSpace**

* Checks the size of the program
* Check is there is enough memory to store/run the program
* If there isn’t enough memory, we exit the program and output a message. This should wake up the parent.
* Have a global variable to keep track of the amount of memory the system contains
* Use a bitmap to keep track of the memory being used
* Redesign the for loop for pageTable to allow it to know the difference between virtual and physical memory
* Output the amount of memory currently being used
* Before the program exits, deallocate the memory

**Task 4: Memory Allocation**

We will implement a -M command line option. This option will be optional. We will allow one integer parameter (1, 2, 3) to decide which algorithm to use for memory allocation. A searching function will be created to search the memory for available space.

1. First-Fit:
   1. This algorithm will take the first found space of available memory that can fit the process, and allocate it for use with the process.
2. Best Fit:
   1. This algorithm will take the smallest found space of available memory that can fit the process, and allocate it for use with the process.
3. Worst Fit:
   1. This algorithm will take the biggest found space of available memory that can fit the process, and allocate it for use with the process.

Search Function:

We will implement this in the bitmap class. The function will take the number of bits needed to store and the allocation method we’re using and iterate through the map, finding any space with enough unset bits to accommodate it. The function will continue to search through until it hits a marked bit, recording the number of open bits. The function will return the index of the space which fits the number of bits according to the given method.

**Task 5: Testing**

Subject our solution to the 3 programs the T.A. presented to us. The 3 programs are as follows: matmult, sort, and halt.

* matmult
  + Test our memory allocation and address space
  + It will show that we would not let a program run if we do not have enough memory
  + Have the ability to allocate and deallocate memory
* sort
  + Test our memory allocation and address space
* halt
  + Test our system call
* other
  + Test invalid files name for execution
  + Test non-executable files for execution
  + Check bad SpaceID for Join()
  + Unprogram system calls
  + Infinite recursion