Segmentation and Paging

CSC 213 – February 11, 2021
Agenda for Today

1. Bug of the Day
2. Assignment and Lab Q&A
3. New Assignment: Archive Printer
4. Address Spaces
5. Paging
6. Wrap Up
Bug of the Day
What's wrong with this code?

```c
int x = 123;
int* y = malloc(sizeof(int));
y = &x;
*y = 321;
```

The code allocates memory using `malloc` but does not free it. This can lead to a memory leak, as shown in the diagram:

- **Locals/Stack**: `x` is on the stack.
- **Heap**: `y` points to a heap-allocated memory block.

The memory allocated by `malloc` is not freed, leading to a memory leak.
Q&A
Uniquelist Assignment Q&A
Worm Lab Q&A

In task_wait, is handle the task we are waiting for?
A: Yes. The task that is trying to wait is current_task.

In task_wait, if handle has not exited yet, should we try to start the waitee?
A: No. If current_task is 3 (just as an example), you should try to run task 4, then 5, etc. even if those aren’t the tasks you’re waiting for. That’s what round robin is.

Can a task wait for multiple tasks at once, or block for more than one reason?
A: No. Once a task blocks, it would have to unblock to call another blocking function.

How do you run the test programs?
A: Change directory (cd) to tests. Run make. Then you should have test1, test2, etc.
   You can run those test programs directly.

Should we call task_sleep inside of task_wait?
A: No, you should just switch to another task that’s ready to run. The one exception to that is that you may have no tasks that are ready to run. Then you might call sleep_ms(10).
How do we run a task?
A: At the start of the program, a task is already running: task 0. When task 0 blocks, you have to run a different task i. Do that by calling swapcontext(&tasks[0].context, &tasks[i].context)

What does swapcontext do?
A: The current path of execution is paused inside of swapcontext, and all the state we need to resume it later is saved into the ucontext_t* passed in as the first parameter. Then, swapcontext loads all the state from the second ucontext_t* and starts executing that one.

In task_wait, if the specified task has already exited can we just return?
A: Yes, you can return immediately. Or you can schedule a new task. Either is fine.

Is it okay if our scheduler code loops over tasks when nothing is runnable?
A: That’s okay. Sleeping is a little more “polite” but it’s not a big issue (no effect on grade)

In the test files, there are lot of red and green underlines. What gives?
A: repl.it is wrong. You can ignore those. I wish it didn’t do that.
Worm Lab Q&A

When do we use a task’s exit context?
A: You don’t have to use it directly. The task_create function sets it up so if the task ever returns from its main function it will then call task_exit using exit_context. You should just leave it there and never swap to it or save to it.

What is the general structure of the worm game that uses our tasks?
A: The main task creates a bunch of tasks to perform important functions for the game. Then it waits for those tasks. The tasks run along as the game proceeds, until you lose. Then the tasks exit, and the main task can resume and shut down.

Can we just assume that a task will never wait for itself?
A: Yeah, that’s okay. I recommend not writing the program the does that.
Assignment: Archive Printer
Address Spaces
What's in a process' address space?

- code
- heap
- stack
- library code
- library data
- program globals
- unused space
- address

$2^{32} - 1$ or $2^{64} - 1$
Segmentation

Process A

- Code
- Heap
- Stack
- Lib

Segment Fault

Process B

- Code
- Heap
- Stack

Physical memory

0 start_ end

XGB start_ end
Paging
What's wrong with segmentation?

Segmentation chops up memory into odd-sized segments.
   If you end up with a bunch of tiny gaps between segments, they may not be usable. We can move a segment over to defragment the gaps between segments, but that copying is costly.

May not work well with multiprocessor systems.
   Probably, in theory, segmentation could work with multiprocessor systems. There weren’t many (or any?) multiprocessor systems that used segmentation, so there might be issues I’m not thinking of.

A segment has to be mapped to a contiguous physical memory range
   Size is limited; one segment cannot be larger than the total size of physical memory. If we need to grow segment, we have to put the newly grown area right next to where the segment already lives, or else we have to copy the entire thing somewhere else. This often leads the OS to waste space by making segments big, and hoping they don’t have to grow.
Alternative Approach: Paging

Virtual Address Space

Physical Memory

One page (4096 bytes)

Page Table

0: unmapped
1: physical page 0, r-x
2: phys page 1, rw-
3: phys page 2, rw-
4: unmapped
5: unmapped
6: phys page 4, rw-
7: phys page 3, rw-
Wrap Up
Reminders

Uniquelist Assignment
Due tonight at 11:59pm

Worm Lab
Due tomorrow at 11:59pm

Read the lab before class on Friday
It's available to read now