Processes:
`sleep()`

Adapted by Joseph Lunderville
from slides by Dr. Brian Fraser
and course material by Dr. Steve Ko

Topics

- What specifically is a **running program?**
- Writing C code to call a **syscall: sleep()**
- Using **man** pages
- Fun with some **C pointers**

Pair Programming

- In lecture, we'll do lots of programming activities!
  - You and a partner will use **Pair Programming**
  - Short video: Pair Programming (by Code.org)

- *Suggestion*
  - Driver typing the code
  - Navigator look up the man page
  - Both are creating the code!

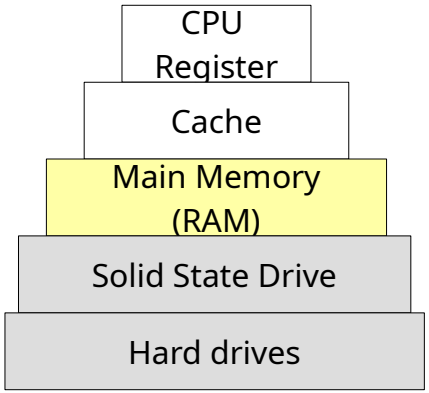
- Short video: ordinary pair programming session (30s)

Process

Process

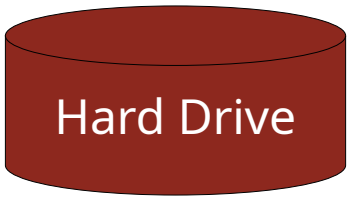
- *What is a **program?***
  - Basically a **compiled executable file**
  - But unless you run it, it’s just a file!
- *What is a **process?***
  - Basically a **running program**
  - It’s not quite that simple, we’ll learn more later

Program In Memory



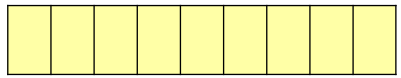
Memory Hierarchy

- CPU can execute instructions from memory
- **Program** (the executable) stored on disk
  - Slow data access (fetch) speed due to distance, spinning drive, etc.
  - CPU cannot access bytes *without loading them into memory*
  - So, a program must be in memory to run

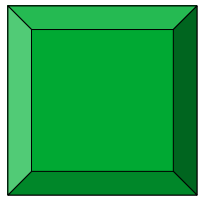


Slow Storage

Data loaded into main memory



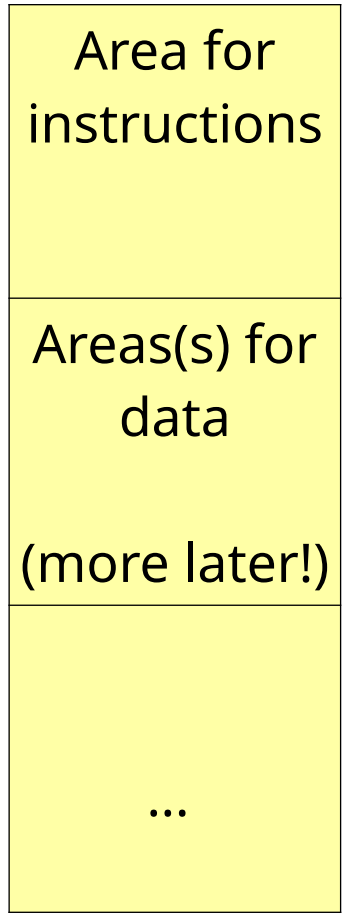
Bytes in Memory:
Fast CPU access



CPU

Start Execution

- *To start executing a program, the OS will*
  - **Create a memory space** in RAM for the program to run
  - **Load the machine code** from the program's file on disk into memory
  - Set up part of memory space for **data** (variables, …) More later!
  - **Start executing** the program from memory (makes it a process!)



Controlling a Process

- *Controlling a process*
  - Programmers use system calls **(syscalls)** to control processes
- *Some core process **syscalls** include*
  - **`fork()`**
    Create a new process by cloning current one
  - **`exec()`**
    Replace current process with another executable
    (a family of different calls serve the same purpose)
  - **`wait()`**
    Wait until a created process finishes its work

Audience Participation - Process

- What is the difference between a **process** and a **program?**

a) A process is a program loaded into memory and running.

b) A program is a process loaded into memory and running.

c) A process is loaded from RAM to the hard drive by the OS.

d) A program is loaded from RAM to the hard drive by the OS.

Coding and Process Activity

Ready to Code

- Open two terminals (tabs or windows)
  - One terminal for coding:
    - Launch the CMPT 201 container
      ```
      > docker start -ai cmpt201
      ```
    - Make a folder for our work
      ```
      > mkdir -p ~lecture/02-forkexecwait
      ```
  - Another terminal to look up **man** pages:
    - Connect to the already running container
      ```
      > docker exec -it cmpt201 zsh —login
      ```
    - Test this out by running
      ```
      > man 3 printf
      ```

If you haven't downloaded the docker image yet, run:

```
> docker create —it —name cmpt201 ghcr.io/sfu-cmpt-201/base
```

Activity - `sleep()`

- Write a program that calls “sleep” in a loop with some timeout (pick a small nonzero number) (5m)
  - Read the **man** page for `sleep()`:
    ```
    > man 3 sleep
    ```
    (“3” specifies the manual section: without it, you get the *shell command* which is also named `sleep`)
- In a third terminal, run `btop`
  - Connect to the running container again using `docker exec...`
  - `btop` is a good tool to visualize parent/child relationships

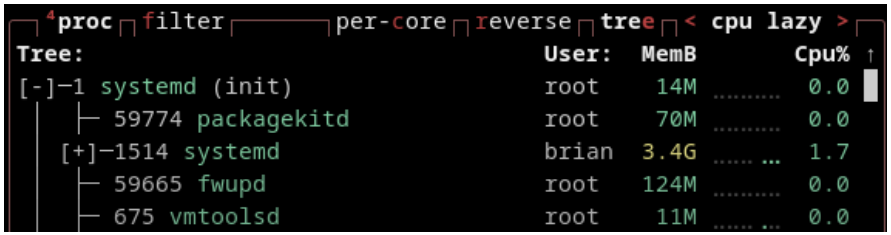
Solution - `sleep()`

- *See process information:* **`btop`**
  - Use tree view (press e)
  - Each process has a parent (except init and kthreadd; not shown in containers).
  - Our container's zsh runs a.out

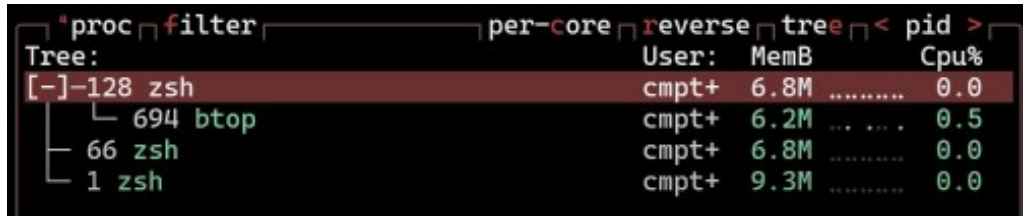
sleep.c > ...

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>


int main()
{
    char* message = "Hello world!\n";
    for (int i = 0; i < strlen(message); i++) {
        printf("%c", message[i]);
        fflush(stdout);
        sleep(2);
    }
    printf("\n");
    printf("DONE\n");
}
```



On Linux shows init



In container, no init

Audience Participation - Docker

- Which command connects to an already running Docker container?
- Which command downloads the Docker container?
- Which command launches the Docker container?

a) `docker start -ai cmpt201`

b) `docker exec -it cmpt201 zsh --login`

c) `docker git clone github.com/sfu-cmpt-201/base`

d) `docker create -it --name cmpt201 ghcr.io/sfu-cmpt-201/base`

Reading a **man** Page

man Pages

- *Reading a man page*
  - Our primary way to learn functions/system calls for systems programming
  - It takes practice to effectively read a man page!
- *The command*

  ```
  man <thing>
  ```

  - e.g., "`man ls`", "`man cd`"
- *Section numbers*
  - Choose between two pages with the same name
  - Most relevant sections for CMPT 201:

    1 – **general** (shell) **commands,** e.g. "`man 1 ls`"

    2 – **system calls,** e.g. "`man 2 fork`"

    3 – **C standard library functions,** e.g. "`man 3 printf`"

Learning a Syscall

- *Problem*
  - I know a syscall; how do I use it?
- *Steps*
  1) Is this **what I want?**
  2) How do I **call it?**
  3) What does it **give me?**
  4) How can it go **wrong?** (**errno, feature test**)

```
atoi(3)                Library Functions Manual                atoi(3)

NAME
       atoi, atol, atoll - convert a string to an integer

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <stdlib.h>

       int atoi(const char *nptr);
       long atol(const char *nptr);
       long long atoll(const char *nptr);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       atoll():
           _ISOC99_SOURCE
               || /* glibc <= 2.19: */ _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION
       The atoi() function converts the initial portion of the
       string pointed to by nptr to int.  The behavior is the
       same as

           strtol(nptr, NULL, 10);

       except that atoi() does not detect errors.

       The atol() and atoll() functions behave the same as
       atoi(), except that they convert the initial portion of
       the string to their return type of long or long long.

RETURN VALUE
       The converted value or 0 on error.
```

Learning a Syscall

1) *Is this what I want?*
   - Read **description** section
   - Skim for relevant part (this is a useful skill!
2) *How do I call it?*
   - Read **synopsis (prototypes)**
   - Check header files and return type
   - Check arguments (in and out)
3) *What does it give me?*
   - Read **return value** section
   - Pay attention to output parameters (pointers)


```
atoi(3)                Library Functions Manual                atoi(3)

NAME
       atoi, atol, atoll - convert a string to an integer

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <stdlib.h>

       int atoi(const char *nptr);
       long atol(const char *nptr);
       long long atoll(const char *nptr);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       atoll():
           _ISOC99_SOURCE
               || /* glibc <= 2.19: */ _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION
       The atoi() function converts the initial portion of the
       string pointed to by nptr to int.  The behavior is the
       same as

           strtol(nptr, NULL, 10);

       except that atoi() does not detect errors.

       The atol() and atoll() functions behave the same as
       atoi(), except that they convert the initial portion of
       the string to their return type of long or long long.

RETURN VALUE
       The converted value or 0 on error.
```

Learning a Syscall

- *How can it go wrong? (errno, feature test)*
  - What errors possible? Read **errors** section (more later)
  - Does it require a feature macro? E.g. `nanosleep()`...

nanosleep():
    _POSIX_C_SOURCE >= 199309L

ERRORS

EFAULT Problem with copying information from user space.

EINTR The pause has been interrupted by a signal that was delivered to the thread (see **signal**(7)). The remaining sleep time has been written into *rem so that the thread can easily call **nanosleep**() again and continue with the pause.

EINVAL The value in the tv_nsec field was not in the range [0, 999999999] or tv_sec was negative.

atoi(3) Library Functions Manual atoi(3)

NAME

atoi, atol, atoll - convert a string to an integer

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <stdlib.h>

int atoi(const char *nptr);
long atol(const char *nptr);
long long atoll(const char *nptr);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

atoll():
    _ISOC99_SOURCE
        || /* glibc <= 2.19: */ _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION

The **atoi**() function converts the initial portion of the string pointed to by nptr to int. The behavior is the same as

```
strtol(nptr, NULL, 10);
```

except that **atoi**() does not detect errors.

The **atol**() and **atoll**() functions behave the same as **atoi**(), except that they convert the initial portion of the string to their return type of long or long long.

RETURN VALUE

The converted value or 0 on error.

Audience Participation - Pointers

- What does this output?

```c
#include <stdio.h>
#include <stdlib.h>

int make_abs_get_product(int *pA, int *pB)
{
    *pA = abs(*pA);
    *pB = abs(*pB);
    return *pA * *pB;
}

int main()
{
    int w = -4;
    int h = 5;
    int area = make_abs_get_product(&w, &h);
    printf("%d x %d = %d\n", w, h, area);
}
```

a) -4 x 5 = -20
b) 4 x 5 = 20
c) 4 x 5 = -20
d) -4 x 5 = 20

Review C Pointers

- *Note* `char** ppdigit`
  - x is a pointer-to-a-pointer
  - Used for **output parameters**
- *Use of* **
  - Calling code passes in the address of their pointer
  - Function sets **where that pointer points**

```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>

bool find_first_digit(char* data, int n, char** ppdigit)
{
    for (int i = 0; i < n; i++) {
        if (isdigit(data[i])) {
            *ppdigit = &data[i];
            return true;
        }
    }
    return false;
}

int main()
{
    char* data = "I wa5 h3r3!\n";
    char* pfirst_digit = NULL;

    if (find_first_digit(data, strlen(data), &pfirst_digit)) {
        printf("Found digit: %c\n", *pfirst_digit);
    } else {
        printf("Found no digits.\n");
    }
}
```

Summary

- *Processes are programs executing from memory (RAM)*
  - Each process has its own **memory space**
- *C Programming*
  - Use **man** pages to lookup functions
  - Pointers and pointers-to-pointers used as output parameters
- *Development Ideas*
  - Use multiple terminal tabs/windows
  - Read documentation first!
  - Code a little at a time: write small experiments
    - Test unfamiliar syscalls, APIs, or data structures in isolation
  - Plan before writing larger chunks
- `sleep()` puts function to sleep