

# Processes: `sleep()`

Based on content  
created by Dr. Steve Ko

# Topics

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

# Pair Programming

- In lecture, we'll do lots of programming activities!
  - You and a partner will use ..
  - Show: **Pair Programming** (by Code.org)
- **Suggestion**
  - Driver typing the code
  - Navigator look up the man page
  - Both are creating the code!
- See **ordinary pair programming session** (show 30s)

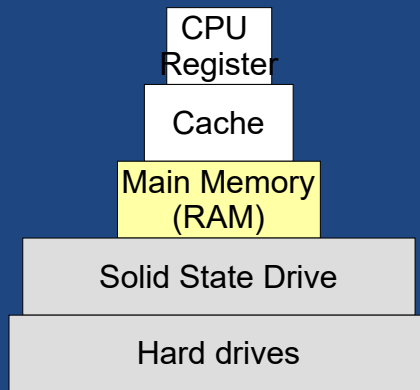


# Process

# Process

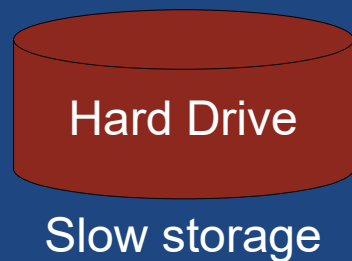
- What is a program?
  - Basically a..
  - But unless you run it, it's just a file!
- What is a process?
  - Basically a..  
(not quite that simple; we'll learn more)

# Program in Memory

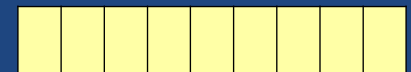


Memory Hierarchy

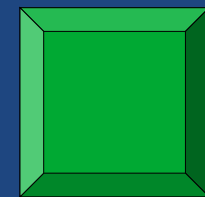
- ..
- Program (the executable) stored on disk.
  - Slow data access (fetch) speed due to distance, spinning drive, etc.
  - CPU cannot access bytes without loading them into memory.
  - So, a program must be in memory to run.



Data loaded into  
main memory



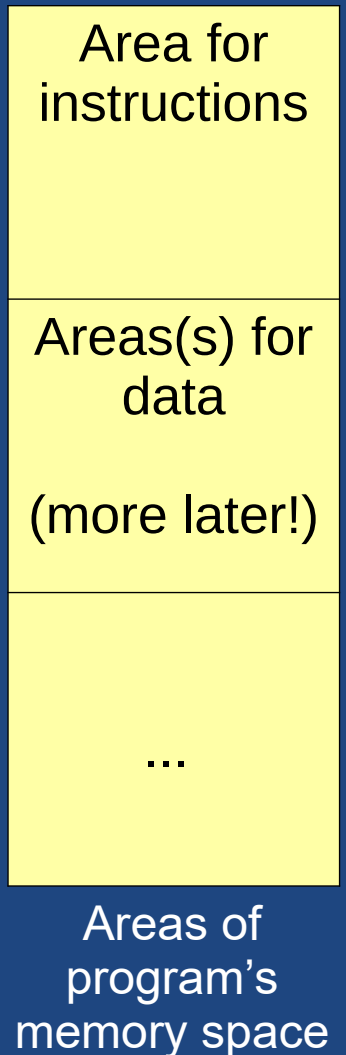
Bytes in Memory:  
Fast CPU access



CPU

# Start Execution

- To start executing a program, the OS will:
  - ..  
in RAM for the program to run
  - load the machine code from the program's file on disk into memory.
  - make 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:
  - ..  
Create a new process by cloning current one.
  - ..  
Replace current process with another executable.  
(family of different calls, but do the same thing).
  - ..  
Wait until a created process finishes its work.



# ABCD: 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 & Process Activity

# Ready to Code

- Open Two Terminals (tabs or windows)
  - A terminal for Coding:
    - Launch the CMPT 201 container:  
`docker start -ai cmpt201`
    - Make a folder for our work  
`mkdir -p ~/lecture/02-forkexecwait`
  - A terminal for 'man' page:
    - connect to the already running container:  
`docker exec -it cmpt201 zsh --login`
    - Run  
`man 3 printf`

If not yet downloaded docker image, first run:

`docker create -it --name cmpt201 ghcr.io/sfu-cmpt-201/base # if needed`

# Activity: Hello C World!

- Create a C program:  
    `cd ~/lecture/02-forkexecwait/  
    nvim hello.c`
- Compile  
    ``clang hello.c``
  - This builds executable `a.out`; run it:  
    `./a.out`
  - Set executable's name:  
    `clang hello.c -o hello`
- (3 mins)  
    You do it now!

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello world!\n");
6  }
```

# Activity: sleep()

- (5 mins)

Write a program that keeps calling `sleep()` with some timeout value.

- Check the man page for `sleep()`:

```
$ man 3 sleep
```

(Without the 3, it will give you the Linux `sleep` command)

- In a 3<sup>rd</sup> terminal, run `bt`

- Connect to running container using `docker exec...`
- `bt` is a good tool to visualize parent/child processes

# sleep() Solution

- See process information: `bt`
  - 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`

```
*proc filter per-core reverse tree < cpu lazy >
Tree:
[ ]-1 systemd (init)
    | 59774 packagekitd
    | [ ]-1514 systemd
    |     | 59665 fwupd
    |     | 675 vmtoolsd
    |     |
    |     | User: MemB Cpu%
    |     |-----
    |     |
    |     | 1
    |     |
    |     | 0.0
    |     |
    |     | 0.0
    |     |
    |     | 1.7
    |     |
    |     | 0.0
    |     |
    |     | 0.0
```

On Linux shows init

```
*proc filter per-core reverse tree < pid >
Tree:
[ ]-128 zsh
    | 694 btop
    | 66 zsh
    | 1 zsh
    |
    | User: MemB Cpu%
    |-----
    |
    | 6.8M
    | 6.2M
    | 6.8M
    | 9.3M
    |
    | 0.0
    | 0.5
    | 0.0
    | 0.0
```

In container, no init

# ABCD: 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 Page

- 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 is `man <da-thing>`
  - e.g., ``man ls``, ``man cd``
- Section Numbers
  - ..
  - Most relevant sections for CMPT 201:
  - `man 1`: General commands e.g., ``man 1 ls``
  - `man 2`: System calls e.g., ``man 2 fork``
  - `man 3`: C standard library functions e.g., ``man 3 printf``

# Learning a Syscall

- Problem
  - I know a syscall;  
how do I use it?
- Steps Overview
  - 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
- ..  
(You'll need this skill!)

## 2) How do I call it?

- Read **Synopsis (prototype)**
- Check header files & 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

## 4) How can it go wrong?

(errno, feature test)

- What errors possible?  
Read **Errors** (more later)
- Do you need to a feature test?

E.g., **man 3 nanosleep**  
must define  
**\_POSIX\_C\_SOURCE**

**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.

# ABCD: Review C Pointers

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int make_abs_get_product(int *pA, int *pB)
5  {
6      *pA = abs(*pA);
7      *pB = abs(*pB);
8      return *pA * *pB;
9  }
10
11 int main()
12 {
13     int w = -4;
14     int h = 5;
15     int area = make_abs_get_product(&w, &h);
16     printf("%d x %d = %d\n", w, h, area);
17 }
```

- What does this output?

a)  $-4 \times 5 = -20$

b)  $4 \times 5 = 20$

c)  $4 \times 5 = -20$

d)  $-4 \times 5 = 20$

(Formatting cleaned up)

# Review C Pointers

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

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  bool find_first_digit(char* data, int n, char** ppdigit)
7  {
8      for (int i = 0; i < n; i++) {
9          if (isdigit(data[i])) {
10             *ppdigit = &data[i];
11             return true;
12         }
13     }
14     return false;
15 }
16
17 int main()
18 {
19     char* data = "I wa5 h3r3!\n";
20     char* pfirst_digit = NULL;
21
22     if (find_first_digit(data, strlen(data), &pfirst_digit)) {
23         printf("Found digit: %c\n", *pfirst_digit);
24     } else {
25         printf("Found no digits.\n");
26     }
27 }
```

# 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
  - Code a little at a time
- sleep() puts function to sleep