Process Environment

Eric Missimer

`main`

- `main` is not the first thing or last thing your program does
- First your program sets up the C *run-time*
- The C run-time is relatively simple, C++ a little more complex, Java run-time really complex
- The code that does this is in `crt0.o`
- `crt0` calls `main`

Process Termination

- Normal Termination:
  - return from `main`
  - Calling `exit`
  - Calling `_exit` or `_Exit`
  - Return of the last thread from its start routine[1]
  - Calling `pthread_exit` from the last thread[1]
- Abnormal Termination:
  - Calling `abort`[2]
  - Receipt of a signal[2]
  - Response of the last thread to a cancellation request[1]

[1] Will discuss later when we talk about threads

[2] Will discuss later when we talk about signals

exit vs. _exit vs. _Exit

- exit
  - All functions registered with atexit are called
    - `int atexit (void (*func)(void))`
  - All C streams are closed
  - All files created with `tmpfile` are removed
  - Control is returned to the host environment

exit vs. _exit vs. _Exit

- `exit`
  - All functions registered with atexit are called
    - `int atexit (void (*func)(void))`
  - All C streams are closed
  - All files created with `tmpfile` are removed
  - Control is returned to the host environment
- `_exit` (syscall)
  - Control is returned to the host environment

exit vs. _exit vs. _Exit

- `exit`
  - All functions registered with atexit are called
    - `int atexit (void (*func)(void))`
  - All C streams are closed
  - All files created with `tmpfile` are removed
  - Control is returned to the host environment
- `_exit` (syscall)
  - Control is returned to the host environment
- `_Exit` (libc)
  - Same as `_exit`
- Code example: `exit.c`

What happens if we return from `main`

- Remember what called `main` for us?

What happens if we return from `main`

- Remember what called `main` for us?
- Answer: `crt0.o`

What happens if we return from `main`

- Remember what called `main` for us?
- Answer: `crt0.o`
- `crt0.o` is written in either assembly or C
- does something like: `exit(main(argc, argv))`

Command-Line Arguments

- We specify the command line arguments via the `exec` syscall
- `exec` is how load a program and run it[3]
- C standard guarantees `argv[argc]` is `null`

[3] We will discuss `exec` in detail in a future class

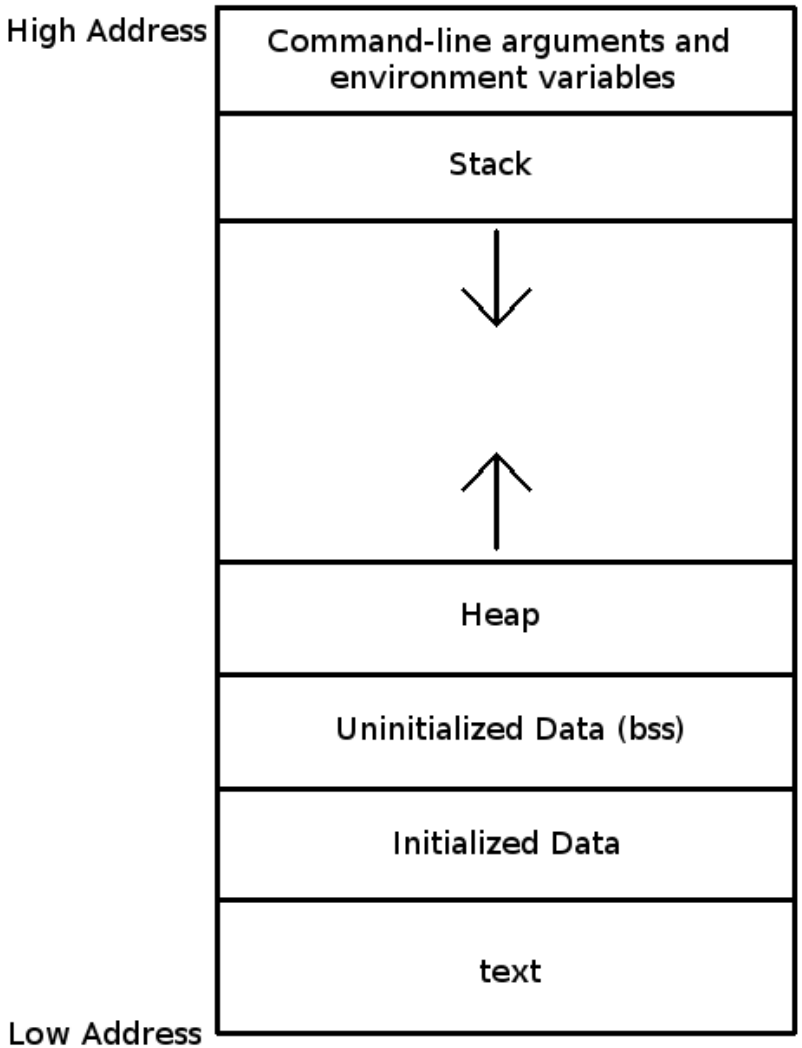
Environment List

- Each process has an environment list:
  `extern char **environ`
- This is a `null`-terminated array of character pointers each pointing to a C string
- Each string is of the form *name*=*value*
- Typical environment variables:
  `HOSTNAME`, `SHELL`, `USER`, `PATH`, `HOME`, `PWD`, `OLDPWD`, `PRINTER`
- run `printenv` from the command line to see environment variables

Environment List

- Environment variables can be set from the command line
  Examples for bash:
  ```
  export VARIABLE=value
  export PATH=$HOME/bin:$PATH
  ```
- `char* getenv(const char* name)` - can be used to retrieve the value of an environment variable
- `int putenv(char *string)` - sets an environment variable, argument string is of the form *name*=*value*
- Can be the third argument to main:
  `int main(int argc, char *argv[], char *envp[])`
  This is available in some Unix systems, but isn't specified by C

Typical Memory Layout



See code examples for stack and heap growing.