Processes II

CS 241

Sept. 25, 2013

exec()

- **fork()**: Duplicates the current process
  - fork() “returns twice”, once as the parent (original) and once as the child process!

exec()

- **exec()**: Executes a file
  - Replaces the current process image with a new process image.

- **exec()** is not a function, but the common name for a family of functions
  - All functions are of the type: **exec_____()**
    - **+“l” (lowercase L): Send arguments as a list.**
    - **+”v”: Send arguments as a vector (array).**
    - **+”e”: Send environmental variables (not used in 241).**
    - **+”p”: Allow searching for the file name.**

main()

- When a new file is executed, the execution begins with the **main()** function.

- Just like in C++
  - void main()
  - int main(int argc, const char *argv[]);

- **Remember**: argv is a NULL terminated array of C-strings!
  - **argv[0]**: Process name
  - **argv[1]**: First command line argument
  - **argv[argc – 1]**: Last command line argument
  - **argv[argc]**: NULL

Example: execlp()

- **execv()**:
  - ~~**+“l” (lowercase L): Send arguments as a list.**~~
  - **+”v”: Send arguments as a vector (array).**
  - ~~**+”e”: Send environmental variables (not used in 241).**~~
  - ~~**+”p”: Allow searching for the file name.**~~

```
int execv(const char *path, char *const arg[]);
```

Example:

```
char *array[] = { "/bin/ls", NULL };
execv("/bin/ls", array);
```

Example #1

```
void main() {
   char *array[] = { "/bin/ls", NULL };




   execv(“/bin/ls", array);




}
```

wait()

- **wait():** Waits for a child process to terminate.
  - **wait()**: Waits for any child process.
  - **waitpid()**: Waits for a specific process.

- A call to wait() retrieves the **exit code** for a process and allows the OS to clean up the process.
  - **exit code**: Value returned from **main()**; integer.
    - **0**: Program finished without error.
    - **Non-0**: Program finished with an error.

Example #2

```
void main() {
   char *array[] = { "/bin/ls", NULL };
   pid_t pid = fork();

   if (pid == 0)
      execv("/bin/ls", array);






}
```

Zombies and Orphans

- A process is a **zombie** if it is a child process of a parent who has not **wait()**’d on it.
  - Zombie processes are still in memory, “wastes” RAM.

- A process is an **orphan** if it’s a child process of a parent that has exited.
  - When a child no longer has a parent, it gets re-parented by the **init process** (pid == 1).