System Programming

Accessing environment variables

- The function `main` supports a third parameter to access environment
- The prototype is

```
int main ( int argc, char ** argv, char ** envp );
```

- The environment variables and their value is passed through `envp`
  - `envp` contains a pointer to the entire environment
  - The format is `var=value`
- You can access it by

```
#include <stdio.h>

// Accessing environment variables

int main ( const int argc, char ** argv, char ** envp )
{
    char ** env_var;
    for ( env_var = envp; *env_var; env_var++ )
        printf ( "%s\n", *env_var );

    return ( 0 );
}
```

- Use `strtok` to separate varable name and value (`env2.c`)
- You can further process the value (`env3.c`)
- A better way to get an individual environment variable is using `getenv` (`getenv.c`)
- You can set an environment variable by using `setenv(3)`

Getting information about a file

- You may need to get a number of details about a file
- Get the size of a file
  - Open the file, go to end of file, find out your location, and return
  - `sz.c`
  - A better way is to use `stat(2)`
    - You can get a lot of information about a file wihout even opening it
  - `sz_stat.c`

Shared memory

- Goals of memory management

  - Privacy: The memory allocated to a process belongs to just that process
  - Isolation: Processes should not be able to access memory that has been allocated to a different process
  - Controlled access: Under certain conditions, a process may be able to provide access to its memory to a different process

- Shared memory
  - Allows for controlled access to a different process
  - Steps to use shared memory
    1. Generate a key to get shared memory using `ftok(3)`
       * Use a pathname and a project identifier to create a System V IPC key
       * The pathname must refer to a file that exists and is accessible to the UID of the process
       * It is a good idea to use a file in the process working directory
       * You can use one of the files in the directory or create a dummy file just for the purpose
    2. Allocate the shared memory segment using `shmget(2)`
       * Get the identifier of the shared memory segment associated with the value of the argument key
       * For flag, use `IPC_CREAT | 0666`
    3. Attach the shared memory in each process that needs to access it
    4. Use shared memory
    5. Detatch shared memory
    6. Release shared memory when the last process using shared memory is terminated
    7. Make sure that shared memory does not linger by using the command `ipcs` from the shell