CS370 - Homework 3

Palindrome Checker

Parallel Processes with Shared Memory and Pipes

Program Description

The purpose of this program will be to process a list of words in parallel using IPC. You will utilize both pipes and shared memory to communicate between the parent and child processes. This program will successfully validate palindromes concurrently and report the results back to the user.

Manager Description

- **Manager** receives n words as command-line arguments
  - ./Manager racecar kayak apple abcba hello
- **Manager** then creates n pipes and n shared memory segments
  - Each child process will have its own pipe and shared memory segment
- For each child process the **Manager** will:
  - Use the pipe to provide the name of the shared memory segment
  - Use shared memory segment to provide the word for the child to check
- **Manager** will launch all child processes *before* waiting
- As the child processes finish the **Manager** will then retrieve the results from the individual shared memory segments and write the results to the console

Palindrome Description

- **Palindrome** will receive a pipe FD (read end) as a command-line argument
- It will then complete a series of steps:
  - ✓ Read the shared memory name from the pipe
  - ✓ Retrieve the assigned word from the struct in the shared memory segment
  - ✓ Check if the word is a palindrome
  - ✓ Write the result (0 or 1) to the result field in the struct

Pipe/Shared Memory

Communication between the manager and each child happens in **two phases**:

Pipe: One-way communication from the manager $\longrightarrow$ child (manager sends the shared memory name to child)

Shared Memory: Two-way access
(manager writes the word $\longleftrightarrow$ child writes back the result)

Parallel Processing

- In Assignment 2, the wait condition for the child was written before the parent process forked the next child.
- This leads to linear/sequential execution. However, for this assignment, we need to execute the processes in parallel.
- Hence, for the concurrent processes the **Manager** must fork the child processes and then use the wait() command for each.

Function Description

- pipe()
- shm_open()
- ftruncate()

- mmap()
- shm_unlink()
- sprintf()

pipe()

Syntax: int pipe(int pipefd[2]);

Arguments: **pipefd**[2] is the array to represent two ends of the pipe. Each end is a file descriptor (FD).

Example:

```
int pipefds[2];
int result_pipe = pipe(pipefds);
```

shm_open()

Syntax: int shm_open(const char *name, int oflag, mode_t mode);

Arguments:

name: name of the memory segment

oflag: can take the following values: O_RDONLY, O_RDWR, O_CREAT, O_EXCL, O_TRUNC

mode: permissions in the form 0666

Example:

char shm_Name[15] = “Shared_Mem0”;

int shm_fd = shm_open(shm_Name, O_CREAT | O_RDWR, 0666);

ftruncate()

Syntax:	int ftruncate(int fd, off_t length);
Arguments:	**fd:** is the file descriptor
	length: is the desired size of the memory segment. (Will be initialized to 0)
Example:	int result = ftruncate(fd, 1234);

mmap()

Syntax: void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

Arguments:

addr: beginning address of the memory object

length: length of the memory object in bytes

prot: protection of the pages (PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE)

flags: Updates to the mapping should be visible to other processes mapping the same region. (MAP_SHARED, MAP_PRIVATE etc.)

mmap()

Arguments: **fd:** returned by shm_open

offset: is 0 in here

Example: mmap(0, size, PROT_READ, MAP_SHARED, shm_fd, 0);

shm_unlink()

Syntax: int shm_unlink(const char *name);

Arguments: **name:** is the memory object name to be unlinked

Example: int error = shm_unlink(shm_Name);

sprintf()

Syntax: int sprintf(char * buffer, const char * string, ...);

Arguments: **string** is stored in **buffer**

Example: sprintf(buffer, "Sum = %d", sum);

Makefile

- Makefile provided, please use it. This is the file we will use to test your program. So, its best if you use it while completing the assignment.

Other Requirements

- Code should compile and run on CS Department computers.
- Submit all .c, along with Makefile and README.txt. **Please remember to submit your assignment in a zipped file.**

Resources

- Read & Write with Pipe
- POSIX Shared Memory

Thank You

If you have any questions, please stop by office hours and we can provide 1on1 assistance!