REPORT

Course - Operating System 1

Instructor - Sathya Peri

What is Happening in the code

Using POSIX shared memory and forking concepts to write a program for calculating Collatz Number in the child process and writing them in shared memory from the child process and accessing the calculated numbers from parent process.

HOW CODE WORKS

1. First a number is taken as input from user for which collatz numbers are to be calculated.
2. Then shared memory is created with read write permissions with **shm_open** function.
3. Then **ftruncate** is used to truncate the created memory to specified size in bytes.
4. Then **mmap** is used to establish a mapping between a process' address space and a file, shared memory object.
5. Then a child process is forked using **fork** and **parent process** is sent to **wait** state.
6. In child process a pointer is created for shared memory.
7. Then Collatz number are calculated and written to shared memory one by one using **sprintf**.

8. After this child stops and parent process continues and prints the data in the Shared memory.
9. After completion Shared Memory is unmapped and closed.

Libraries Used

1.	sys/stat	-	For mode constraints
2.	Fcntl	-	For O_ constraints
3.	sys/wait	-	Wait for process to change state
4.	Unistd	-	Provides access to the POSIX operating system API.
5.	sys/mman	-	Map or Unmap files or devices into memory

Shm_open

shm_open() creates and opens a new, or opens an existing, POSIX shared memory object. A POSIX shared memory object is in effect a handle which can be used by unrelated processes to mmap(2) the same region of shared memory.
General Format - **int shm_open(const char ****name***, int** ***oflag***, mode_t** ***mode*****);**

name - shared memory object to be created or to be opened.

oflag - is a bit mask created by ORing together exactly one of O_RDONLY or O_RDWR with other flags such as O_RDONLY,O_RDWR,O_CREAT,O_EXCL,O_TRUNC.

mode - file access modes such as 0666, 0600, 0621 etc.

Ftruncate

The ftruncate() function truncates the file indicated by the open file descriptor file_descriptor to the indicated length. file_descriptor must be a "regular file" that is open for writing. If the file size exceeds length, any extra data is discarded. If the file size is smaller than length, the file is extended and filled with binary zeros to the indicated length.The ftruncate() function does not modify the current file offset for any open file descriptors associated with the file.

General Format - int ftruncate(int file_descriptor, off_t length);

File_descriptor - (Input) The file descriptor of the file.

Length - (Input) The desired size of the file in bytes.

Mmap

mmap() creates a new mapping in the virtual address space of calling process. The starting address for the new mapping is in addr. The length argument specifies the length of mapping. If *addr* is NULL, then the kernel chooses the address at which create the mapping; this is the most portable method of creating a mapping. If *addr* is not NULL, then the kernel takes it as a hint where to place the mapping; on Linux, the mapping will be at a nearby page boundary. The address of the new mapping is as the result of the call.

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

Parameters

Addr	-	The starting address of the memory area to be mapped.
Length	-	The length in bytes to map.
Prot	-	The access allowed to this process for this mapping i.e. Read/Write ,Read , Write etc
Flags	-	Further defines the type of mapping desired.(MAP_PRIVATE/SHARED)

MAP_PRIVATE option will cause a copy on write mapping to be created. A change to the mapping results in a change to a private copy of the affected portion of the file. These changes cannot be seen by other processes.
MAP_SHARED option provides a memory mapping of the file where changes (if allowed by the protection parameter) are made to the file. Changes are shared with other processes when MAP_SHARED is specified.

Fd	-	An open file descriptor.
Offset	-	The offset into the file, in bytes, where the map should begin.

Fork

Creates a new process. The new process (the child process) is an exact duplicate of the process that calls fork() (the parent process).

pid_t fork(void);

Return Type

Negative Value: creation of a child process was unsuccessful.
Zero: Returned to the newly created child process.
Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

Wait

Suspends the calling process until any one of its child processes ends. More precisely, wait() suspends the calling process until the system obtains status information on the ended child. If the system already has status information on a completed child process when wait() is called, wait() returns immediately. wait() is also ended if the calling process receives a signal whose action is either to execute a signal handler or to end the process.

```
pid_t wait(int *status_ptr);
```

The argument status_ptr points to a location where wait() can store a status value. This status value is zero if the child process explicitly returns zero status. If it is not zero, it can be analyzed with the status analysis macros.

Sprintf

int sprintf(char *buffer, const char *format-string, argument-list);

The sprintf() function formats and stores a series of characters and values in the array buffer. Any argument-list is converted and put out according to the corresponding format specification in the format-string.
The format-string consists of ordinary characters and has the same form and function as the format-string argument for the printf() function.

Return Value

The sprintf() function returns the number of bytes that are written in the array, not counting the ending null character.

Algorithm For Calculating Collatz Number

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod{2} \\ 3n+1 & \text{if } n \equiv 1 \pmod{2}. \end{cases}$$

Analysis Of Output

The end of collatz number function is always number 1 .
Maximum number of steps for a number under 100 billion is 1228 steps for 75,128,138,247.

References

Man Pages and OPERATING SYSTEM CONCEPTS by ABRAHAM SILBERSCHATZ