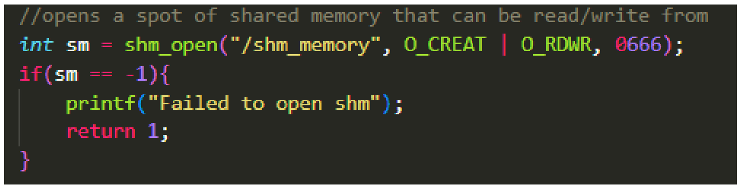
Gavin Ward

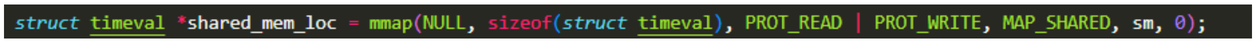
CWID: 12368553

The program starts by creating shared memory through the use of shm_open that allow multiple processes to communicate.



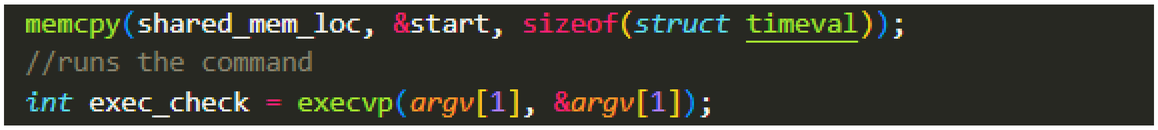
```
//opens a spot of shared memory that can be read/write from
int sm = shm_open("/shm_memory", O_CREAT | O_RDWR, 0666);
if(sm == -1){
    printf("Failed to open shm");
    return 1;
}
```

The shm_open function takes three parameters, the name (/shm_memory), the settings, and the permissions. This is done so that the parent and child can communicate through the shared memory space. Without creating a shared memory space we would not be able to save the start time for later access in the child process and access it in the parent process. For such communications it is necessary to have a shared memory and can be invaluable when intercommunication between different processes is necessary. To access this shared memory location we need to map it to a pointer which we can use to read/write to the shared memory. This is done through the mmap() function where we pass the size, protections, flags, and file descriptor from shm_open.



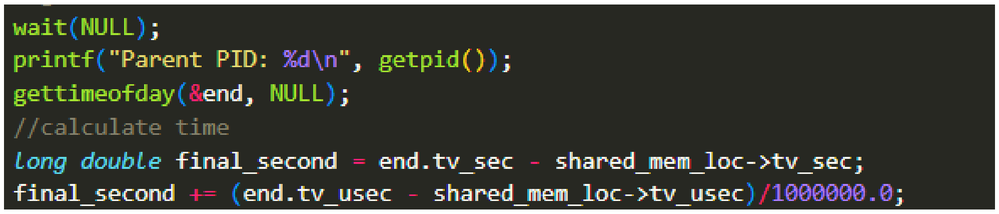
```
struct timeval *shared_mem_loc = mmap(NULL, sizeof(struct timeval), PROT_READ | PROT_WRITE, MAP_SHARED, sm, 0);
```

From there we create a fork which makes a child process that runs the command. The time the child process is entered is saved to the shared memory to be accessed by the parent process later. This can be seen in the screenshot below.



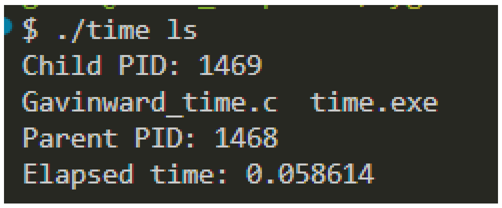
```
memcpy(shared_mem_loc, &start, sizeof(struct timeval));
//runs the command
int exec_check = execvp(argv[1], &argv[1]);
```

While the child process is executing the parent process is in a waiting state due to the wait(NULL) and once the child is finished executing the parent process will begin executing again. The parent process notes the end time and pulls the start time from the shared memory to calculate the total elapsed time.



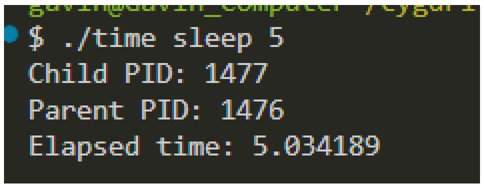
```
wait(NULL);
printf("Parent PID: %d\n", getpid());
gettimeofday(&end, NULL);
//calculate time
long double final_second = end.tv_sec - shared_mem_loc->tv_sec;
final_second += (end.tv_usec - shared_mem_loc->tv_usec)/1000000.0;
```

Finally the program cleans the shared memory and ends. When we pass the ls command as one of the arguments when we run the program we get this output:



```
$ ./time ls
Child PID: 1469
Gavinward_time.c  time.exe
Parent PID: 1468
Elapsed time: 0.058614
```

This output has the child and parent PID, the output of the ls command, and the elapsed time. From this we can see that the program runs the ls command properly within the child process and runs In the expected time. To further illustrate the elapsed time, we can run the program with the sleep 5 command as the arguments. This command will sleep and not produce any output for 5 seconds and then continue execution. Below is the output it produced.



```
$ ./time sleep 5
Child PID: 1477
Parent PID: 1476
Elapsed time: 5.034189
```

As we can see from this it takes the program nearly 5 seconds to execute showing that the program accurately measure the time it takes to run commands.