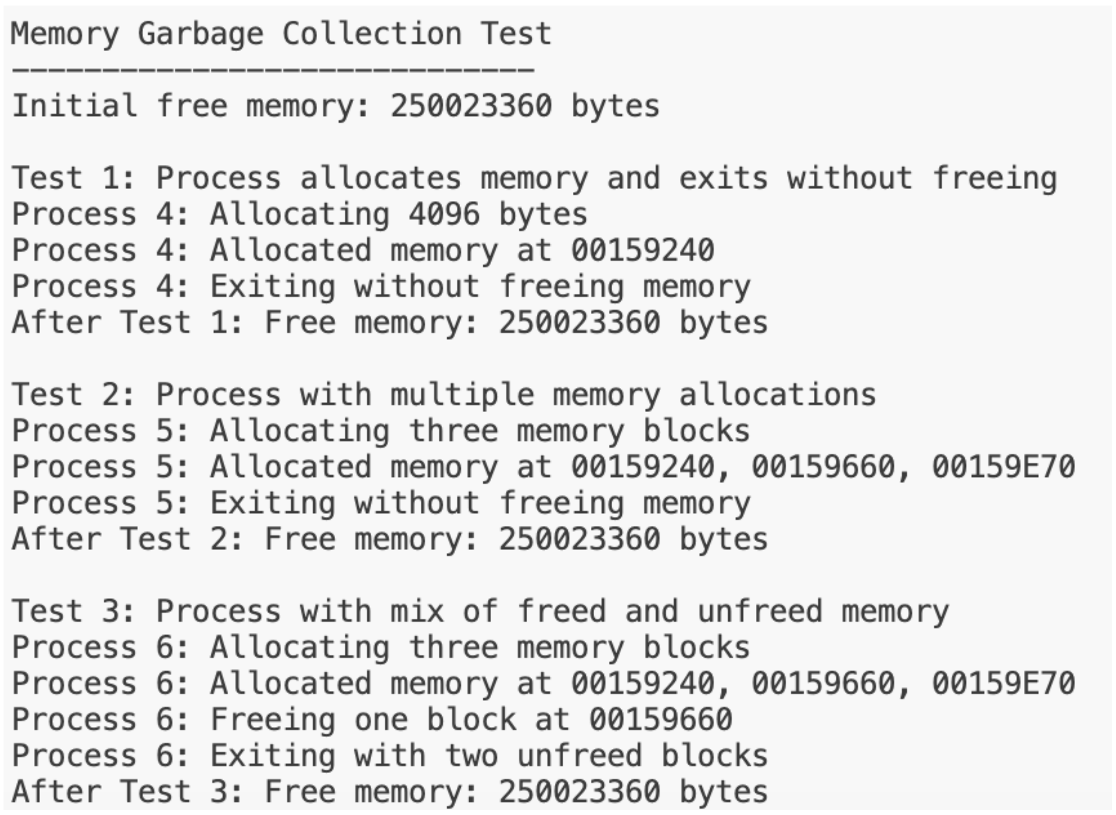
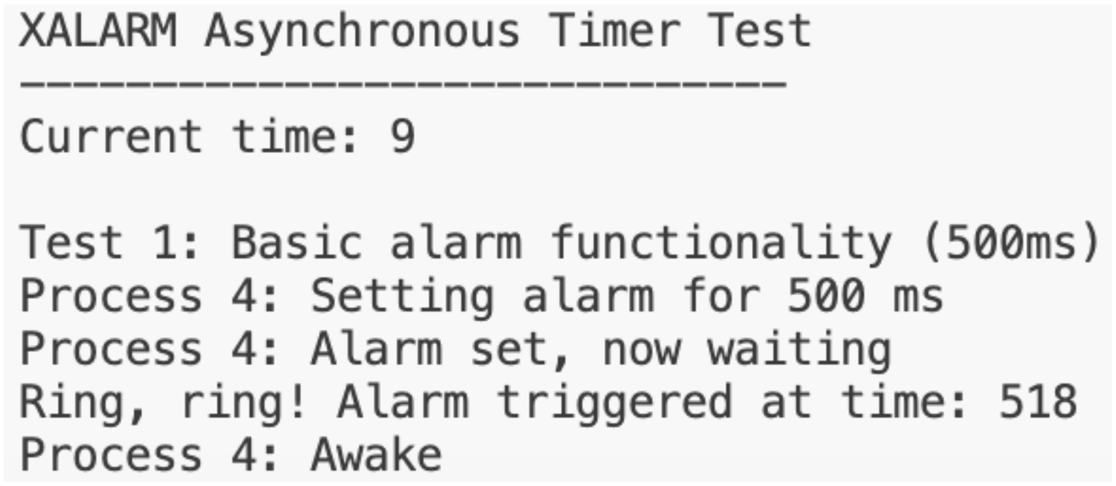
3. We tested multiple cases, including a single memory allocation without freeing, multiple memory allocations without freeing, and multiple memory allocations with one of them freed. Unfreed memory was successfully reclaimed when processes terminated in all 3 cases. See the test results in Figure 1.



```
Memory Garbage Collection Test
------------------------------
Initial free memory: 250023360 bytes

Test 1: Process allocates memory and exits without freeing
Process 4: Allocating 4096 bytes
Process 4: Allocated memory at 00159240
Process 4: Exiting without freeing memory
After Test 1: Free memory: 250023360 bytes

Test 2: Process with multiple memory allocations
Process 5: Allocating three memory blocks
Process 5: Allocated memory at 00159240, 00159660, 00159E70
Process 5: Exiting without freeing memory
After Test 2: Free memory: 250023360 bytes

Test 3: Process with mix of freed and unfreed memory
Process 6: Allocating three memory blocks
Process 6: Allocated memory at 00159240, 00159660, 00159E70
Process 6: Freeing one block at 00159660
Process 6: Exiting with two unfreed blocks
After Test 3: Free memory: 250023360 bytes
```

Figure 1

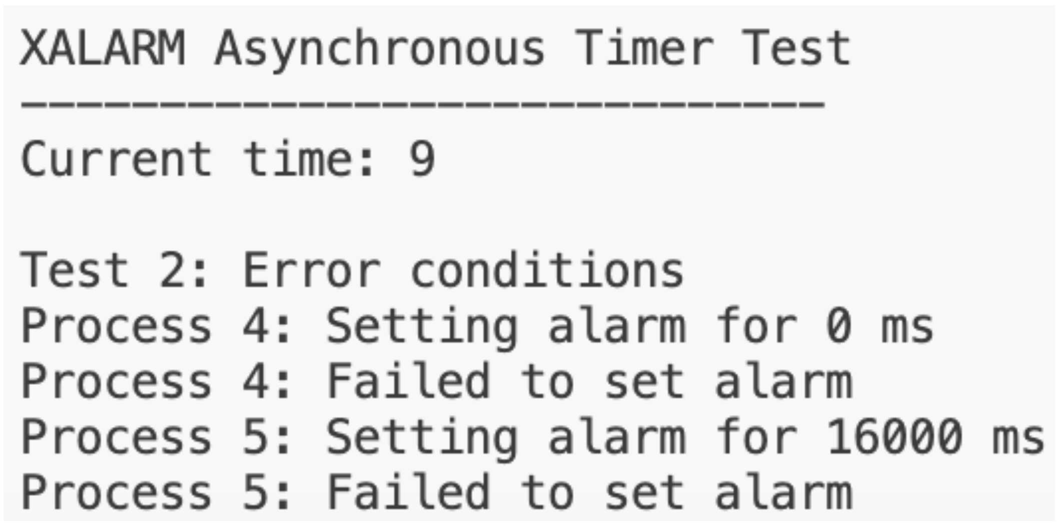
4. We first test the basic alarm functionality and verified that it runs correctly. See the test result in Figure 2.



```
XALARM Asynchronous Timer Test
-----------------------------
Current time: 9

Test 1: Basic alarm functionality (500ms)
Process 4: Setting alarm for 500 ms
Process 4: Alarm set, now waiting
Ring, ring! Alarm triggered at time: 518
Process 4: Awake
```

Figure 2

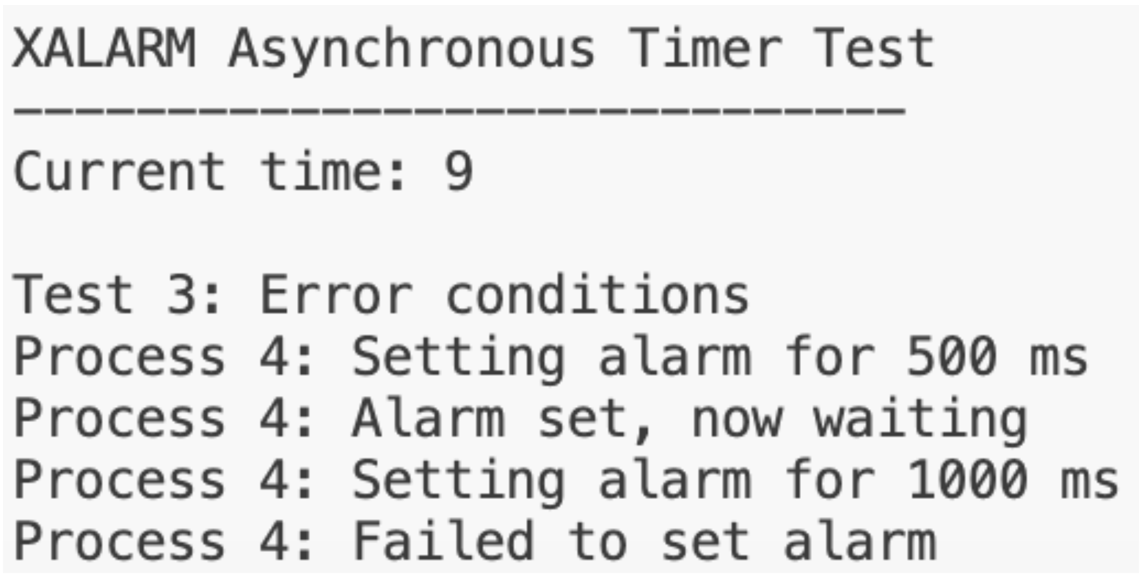
Then, we set alarms with invalid time. Our implementation returned STSERR as expected, see the test results in Figure 3.



```
XALARM Asynchronous Timer Test
------------------------------
Current time: 9

Test 2: Error conditions
Process 4: Setting alarm for 0 ms
Process 4: Failed to set alarm
Process 5: Setting alarm for 16000 ms
Process 5: Failed to set alarm
```

Figure 3

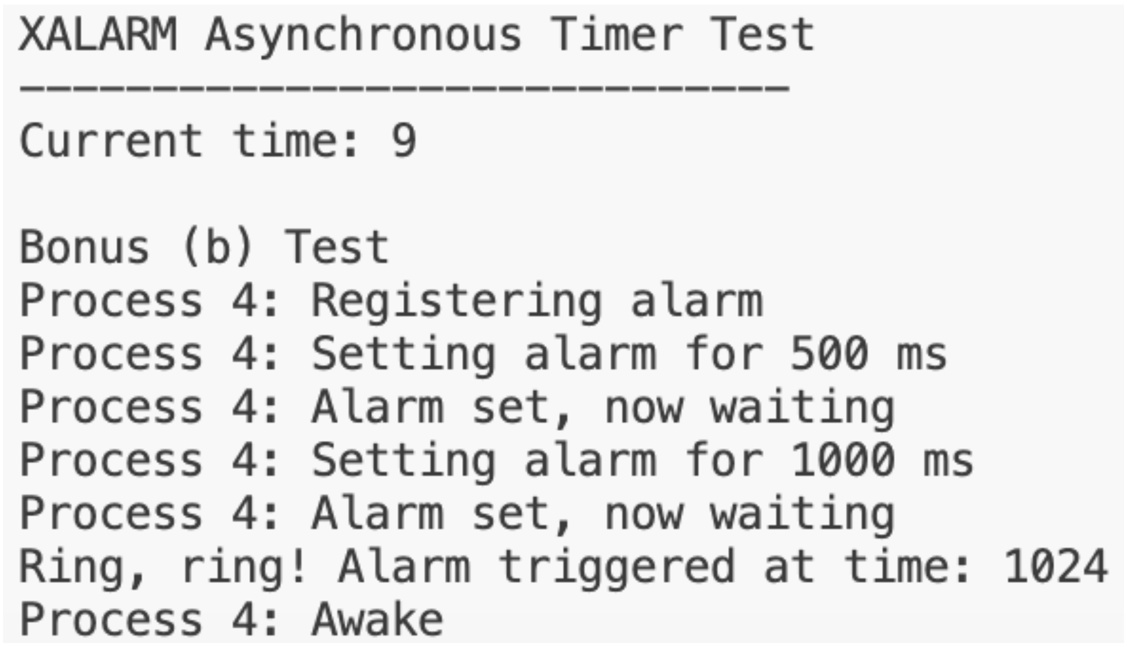
Finally, we set more than one alarms in one process. Since it is stated that “we will disallow xinualarm() being called twice”, our implementation should also return SYSERR in this case. See the test result in Figure 4.



```
XALARM Asynchronous Timer Test
------------------------------
Current time: 9

Test 3: Error conditions
Process 4: Setting alarm for 500 ms
Process 4: Alarm set, now waiting
Process 4: Setting alarm for 1000 ms
Process 4: Failed to set alarm
```

Figure 4

Bonus (b). We implemented xalrmreg() and xalrmset() to allow xalrmset() to be called multiple times where the most recent call overwrites pralrmcounter set by previous calls. We run the test and the alarm was updated successfully. See the test results in Figure 5.



```
XALARM Asynchronous Timer Test
------------------------------
Current time: 9

Bonus (b) Test
Process 4: Registering alarm
Process 4: Setting alarm for 500 ms
Process 4: Alarm set, now waiting
Process 4: Setting alarm for 1000 ms
Process 4: Alarm set, now waiting
Ring, ring! Alarm triggered at time: 1024
Process 4: Awake
```

Figure 5