

# ECS 408/608: Operating System

## Assignment: Critical Section Problem

March 29, 2025

Rohan Mehra  
(Roll No: 21224)

---

1. Consider a system with one parent process and two child processes, C1 and C2. There is a shared integer X initialized to 0. Process C1 increments X by two 10 times in a for loop. Meanwhile, process C2 decrements X by two 10 times in a for loop. After both C1 and C2 are finished, the parent process prints out the final value of X. To solve this, declare a shared memory variable to hold X (see the calls `shmget()`, `shmat()`, `shmdt()`, and `shmctl` in Linux). Write the programs for processes C1 and C2. Ensure you do not place any synchronization code in the code for C1 and C2. You should write the code in such a way that you can simulate race conditions in your program by slowing down C1 or C2 appropriately by using `sleep()` calls at appropriate points. Note that if there is no race condition, the value of X finally should be 0. Simulating race condition means that if you run the program a few times, sometimes the final value of X printed by your program should be non-zero.

**Answer:** The problem involves simulating race conditions in a parent process with two child processes, C1 and C2, interacting with a shared variable X. Initially,  $X = 0$ . Process C1 increments X by 2 ten times, while C2 decrements X by 2 ten times. The parent process prints the final value of X.

**Solution Approach:** To simulate race conditions:

- Used `sleep()` calls in C1 and C2 to introduce delays.
- Ensured no synchronization mechanisms were used, allowing interleaving of operations on X.
- Observed that the final value of X sometimes deviates from 0 due to race conditions.

**Expected Output:** Without proper synchronization, the final value of X is non-deterministic and may vary across runs. For example:

- Expected value without race conditions:  $X = 0$ .
- Observed values with race conditions:  $X \neq 0$  (e.g.,  $X = 4$ ,  $X = -2$ , etc.).

**Code:** The implementation can be found in the GitHub repository: [GitHub Repo](#).

```

PS C:\Users\Rohan Mehra> g++ Critical-Section-Problem-11.cpp -o Critical-Section-Problem-11.exe
PS C:\Users\Rohan Mehra> .\Critical-Section-Problem-11
C1 incremented X to: 2
C2 decremented X to: -2
C2 decremented X to: -4
C1 incremented X to: 0
C2 decremented X to: -2
C1 incremented X to: 2
C1 incremented X to: 0
C2 decremented X to: -4
C1 incremented X to: -2
C2 decremented X to: -6
C1 incremented X to: -4
C2 decremented X to: -8
C1 incremented X to: -6
C2 decremented X to: -10
C2 decremented X to: -12
C1 incremented X to: -8
C1 incremented X to: -6
C2 decremented X to: -10
C2 decremented X to: C1 incremented X to: -4
-8
Final value of X: -4
PS C:\Users\Rohan Mehra>
  
```

2. Add synchronization code based on semaphores to process C1 and C2 above so that race conditions are not possible. Use the call `semget()`, `semop()`, `semctl()` in linux to create and manage semaphore.

**Answer:** The problem involves resolving race conditions in a parent process with two child processes, *C1* and *C2*, interacting with a shared variable *X*. Initially,  $X = 0$ . Process *C1* increments *X* by 2 ten times, while *C2* decrements *X* by 2 ten times. The goal is to ensure that the final value of *X* is always 0, regardless of interleaving.

**Solution Approach:** To eliminate race conditions:

- Used semaphores to synchronize access to the shared variable *X*
- Ensured mutual exclusion by allowing only one process (*C1* or *C2*) to modify *X* at a time
- Implemented the synchronization using the Windows API functions `CreateSemaphore` and `ReleaseSemaphore`

- The final value of  $X$  is deterministic and always equals 0
- No race conditions occur, ensuring consistent behavior across runs

**Code:** The implementation can be found in the GitHub repository: [GitHub Repo](#).

The screenshot displays a Windows desktop environment. In the foreground, a Windows PowerShell window is open, showing the execution of a C++ program. The command prompt shows the user running the command `g++ Critical-Section-Problem-21.cpp -o Critical-Section-Problem-21.exe` and then `./Critical-Section-Problem-21`. The output of the program is visible, showing the execution of the program and the final value of X.

```

C:\Users>
1 #include <iostream>
2 #include <semaphore.h>
3 #include <unistd.h>
4 #include <pthread.h>
5
6 // Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
7
8 // PS C:\Users\Rohan Mehra> g++ Critical-Section-Problem-21.cpp -o Critical-Section-Problem-21.exe
9 // PS C:\Users\Rohan Mehra> .\Critical-Section-Problem-21
10 // C1 incremented X to: 0
11 // C1 incremented X to: 0
12 // C2 decremented X to: 0
13 // C1 incremented X to: 0
14 // C2 decremented X to: 0
15 // C1 incremented X to: 0
16 // C2 decremented X to: 0
17 // C1 incremented X to: 0
18 // C2 decremented X to: 0
19 // C1 incremented X to: 0
20 // C2 decremented X to: 0
21 // C1 incremented X to: 0
22 // C2 decremented X to: 0
23 // C1 incremented X to: 0
24 // C2 decremented X to: 0
25 // C1 incremented X to: 0
26 // C2 decremented X to: 0
27 // C1 incremented X to: 0
28 // C2 decremented X to: 0
29 // Final value of X: 0
30 // PS C:\Users\Rohan Mehra>
31
32 // C1 incremented X to: 0
33 // C2 decremented X to: 0
34 // C1 incremented X to: 0
35 // C2 decremented X to: 0
36 // Final value of X: 0
37
38 // C1 incremented X to: 0
39 // C2 decremented X to: 0
40 // C1 incremented X to: 0
41 // C2 decremented X to: 0
42 // Final value of X: 0
43
44 // C1 incremented X to: 0
45 // C2 decremented X to: 0
46 // C1 incremented X to: 0
47 // C2 decremented X to: 0
48 // Final value of X: 0
49
50 // C1 incremented X to: 0
51 // C2 decremented X to: 0
52 // C1 incremented X to: 0
53 // C2 decremented X to: 0
54 // Final value of X: 0
55
56 // C1 incremented X to: 0
57 // C2 decremented X to: 0
58 // C1 incremented X to: 0
59 // C2 decremented X to: 0
60 // Final value of X: 0
61
62 // C1 incremented X to: 0
63 // C2 decremented X to: 0
64 // C1 incremented X to: 0
65 // C2 decremented X to: 0
66 // Final value of X: 0
67
68 // C1 incremented X to: 0
69 // C2 decremented X to: 0
70 // C1 incremented X to: 0
71 // C2 decremented X to: 0
72 // Final value of X: 0
73
74 // C1 incremented X to: 0
75 // C2 decremented X to: 0
76 // C1 incremented X to: 0
77 // C2 decremented X to: 0
78 // Final value of X: 0
79
80 // C1 incremented X to: 0
81 // C2 decremented X to: 0
82 // C1 incremented X to: 0
83 // C2 decremented X to: 0
84 // Final value of X: 0
85
86 // C1 incremented X to: 0
87 // C2 decremented X to: 0
88 // C1 incremented X to: 0
89 // C2 decremented X to: 0
90 // Final value of X: 0
91
92 // C1 incremented X to: 0
93 // C2 decremented X to: 0
94 // C1 incremented X to: 0
95 // C2 decremented X to: 0
96 // Final value of X: 0
97
98 // C1 incremented X to: 0
99 // C2 decremented X to: 0
100 // C1 incremented X to: 0
101 // C2 decremented X to: 0
102 // Final value of X: 0
103
104 // C1 incremented X to: 0
105 // C2 decremented X to: 0
106 // C1 incremented X to: 0
107 // C2 decremented X to: 0
108 // Final value of X: 0
109
110 // C1 incremented X to: 0
111 // C2 decremented X to: 0
112 // C1 incremented X to: 0
113 // C2 decremented X to: 0
114 // Final value of X: 0
115
116 // C1 incremented X to: 0
117 // C2 decremented X to: 0
118 // C1 incremented X to: 0
119 // C2 decremented X to: 0
120 // Final value of X: 0
121
122 // C1 incremented X to: 0
123 // C2 decremented X to: 0
124 // C1 incremented X to: 0
125 // C2 decremented X to: 0
126 // Final value of X: 0
127
128 // C1 incremented X to: 0
129 // C2 decremented X to: 0
130 // C1 incremented X to: 0
131 // C2 decremented X to: 0
132 // Final value of X: 0
133
134 // C1 incremented X to: 0
135 // C2 decremented X to: 0
136 // C1 incremented X to: 0
137 // C2 decremented X to: 0
138 // Final value of X: 0
139
140 // C1 incremented X to: 0
141 // C2 decremented X to: 0
142 // C1 incremented X to: 0
143 // C2 decremented X to: 0
144 // Final value of X: 0
145
146 // C1 incremented X to: 0
147 // C2 decremented X to: 0
148 // C1 incremented X to: 0
149 // C2 decremented X to: 0
150 // Final value of X: 0
151
152 // C1 incremented X to: 0
153 // C2 decremented X to: 0
154 // C1 incremented X to: 0
155 // C2 decremented X to: 0
156 // Final value of X: 0
157
158 // C1 incremented X to: 0
159 // C2 decremented X to: 0
160 // C1 incremented X to: 0
161 // C2 decremented X to: 0
162 // Final value of X: 0
163
164 // C1 incremented X to: 0
165 // C2 decremented X to: 0
166 // C1 incremented X to: 0
167 // C2 decremented X to: 0
168 // Final value of X: 0
169
170 // C1 incremented X to: 0
171 // C2 decremented X to: 0
172 // C1 incremented X to: 0
173 // C2 decremented X to: 0
174 // Final value of X: 0
175
176 // C1 incremented X to: 0
177 // C2 decremented X to: 0
178 // C1 incremented X to: 0
179 // C2 decremented X to: 0
180 // Final value of X: 0
181
182 // C1 incremented X to: 0
183 // C2 decremented X to: 0
184 // C1 incremented X to: 0
185 // C2 decremented X to: 0
186 // Final value of X: 0
187
188 // C1 incremented X to: 0
189 // C2 decremented X to: 0
190 // C1 incremented X to: 0
191 // C2 decremented X to: 0
192 // Final value of X: 0
193
194 // C1 incremented X to: 0
195 // C2 decremented X to: 0
196 // C1 incremented X to: 0
197 // C2 decremented X to: 0
198 // Final value of X: 0
199
200 // C1 incremented X to: 0
201 // C2 decremented X to: 0
202 // C1 incremented X to: 0
203 // C2 decremented X to: 0
204 // Final value of X: 0
205
206 // C1 incremented X to: 0
207 // C2 decremented X to: 0
208 // C1 incremented X to: 0
209 // C2 decremented X to: 0
210 // Final value of X: 0
211
212 // C1 incremented X to: 0
213 // C2 decremented X to: 0
214 // C1 incremented X to: 0
215 // C2 decremented X to: 0
216 // Final value of X: 0
217
218 // C1 incremented X to: 0
219 // C2 decremented X to: 0
220 // C1 incremented X to: 0
221 // C2 decremented X to: 0
222 // Final value of X: 0
223
224 // C1 incremented X to: 0
225 // C2 decremented X to: 0
226 // C1 incremented X to: 0
227 // C2 decremented X to: 0
228 // Final value of X: 0
229
230 // C1 incremented X to: 0
231 // C2 decremented X to: 0
232 // C1 incremented X to: 0
233 // C2 decremented X to: 0
234 // Final value of X: 0
235
236 // C1 incremented X to: 0
237 // C2 decremented X to: 0
238 // C1 incremented X to: 0
239 // C2 decremented X to: 0
240 // Final value of X: 0
241
242 // C1 incremented X to: 0
243 // C2 decremented X to: 0
244 // C1 incremented X to: 0
245 // C2 decremented X to: 0
246 // Final value of X: 0
247
248 // C1 incremented X to: 0
249 // C2 decremented X to: 0
250 // C1 incremented X to: 0
251 // C2 decremented X to: 0
252 // Final value of X: 0
253
254 // C1 incremented X to: 0
255 // C2 decremented X to: 0
256 // C1 incremented X to: 0
257 // C2 decremented X to: 0
258 // Final value of X: 0
259
260 // C1 incremented X to: 0
261 // C2 decremented X to: 0
262 // C1 incremented X to: 0
263 // C2 decremented X to: 0
264 // Final value of X: 0
265
266 // C1 incremented X to: 0
267 // C2 decremented X to: 0
268 // C1 incremented X to: 0
269 // C2 decremented X to: 0
270 // Final value of X: 0
271
272 // C1 incremented X to: 0
273 // C2 decremented X to: 0
274 // C1 incremented X to: 0
275 // C2 decremented X to: 0
276 // Final value of X: 0
277
278 // C1 incremented X to: 0
279 // C2 decremented X to: 0
280 // C1 incremented X to: 0
281 // C2 decremented X to: 0
282 // Final value of X: 0
283
284 // C1 incremented X to: 0
285 // C2 decremented X to: 0
286 // C1 incremented X to: 0
287 // C2 decremented X to: 0
288 // Final value of X: 0
289
290 // C1 incremented X to: 0
291 // C2 decremented X to: 0
292 // C1 incremented X to: 0
293 // C2 decremented X to: 0
294 // Final value of X: 0
295
296 // C1 incremented X to: 0
297 // C2 decremented X to: 0
298 // C1 incremented X to: 0
299 // C2 decremented X to: 0
300 // Final value of X: 0
301
302 // C1 incremented X to: 0
303 // C2 decremented X to: 0
304 // C1 incremented X to: 0
305 // C2 decremented X to: 0
306 // Final value of X: 0
307
308 // C1 incremented X to: 0
309 // C2 decremented X to: 0
310 // C1 incremented X to: 0
311 // C2 decremented X to: 0
312 // Final value of X: 0
313
314 // C1 incremented X to: 0
315 // C2 decremented X to: 0
316 // C1 incremented X to: 0
317 // C2 decremented X to: 0
318 // Final value of X: 0
319
320 // C1 incremented X to: 0
321 // C2 decremented X to: 0
322 // C1 incremented X to: 0
323 // C2 decremented X to: 0
324 // Final value of X: 0
325
326 // C1 incremented X to: 0
327 // C2 decremented X to: 0
328 // C1 incremented X to: 0
329 // C2 decremented X to: 0
330 // Final value of X: 0
331
332 // C1 incremented X to: 0
333 // C2 decremented X to: 
```

3. In this program, you will write a program to solve the m-producer n- consumer problem, m, n greater than equal to 1. You have a shared circular buffer that can hold 20 integers. Each producer process stores the numbers 1 to 100 in the buffer one by one (in a for loop with 100 iterations) and then exits. Each consumer process reads the number from the buffer and adds them to a shared variable SUM(initialized to 0). Any consumer process can read any of the numbers in the buffer. The only constraint is that the consumers should read every number written by some producer exactly once. However, a producer should not write when the buffer is full; a consumer should not read when the buffer is empty.

Write a program that creates the shared circular buffer and the shared variable SUM using the shm\*() calls in Linux. You can create any other shared variable that you may need. The program then reads in the value of m and n from the user and forks m producers and n consumers. The producer and consumer codes can be written as functions that are called by the child processes. After all the producers and

consumers have finished (the consumers exit after all the data produced by all the producers have been read. How does a consumer know this?), the parent process prints the value of SUM. Test your program with at least (a)  $m=1, n=1$ , (b)  $m=1, n=2$ , (c)  $m=2, n=1$ , and (d)  $m=2, n=2$ .

**Answer:**

To solve the problem:

- Used a shared circular buffer with synchronization mechanisms to manage access between producers and consumers.
- Implemented semaphores (`emptySemaphore`, `fullSemaphore`) to track empty and filled slots in the buffer.
- Used a mutex (`BufferMutex`) to ensure mutual exclusion when accessing the shared buffer and indices.
- Maintained counters (`produced_count`, `consumed_count`) to track the total numbers produced and consumed.
- Consumers exit only after all producers have finished and all produced numbers have been consumed.

**Expected Output:** With proper synchronization:

- The final value of SUM equals the sum of all numbers produced by  $m$  producers:

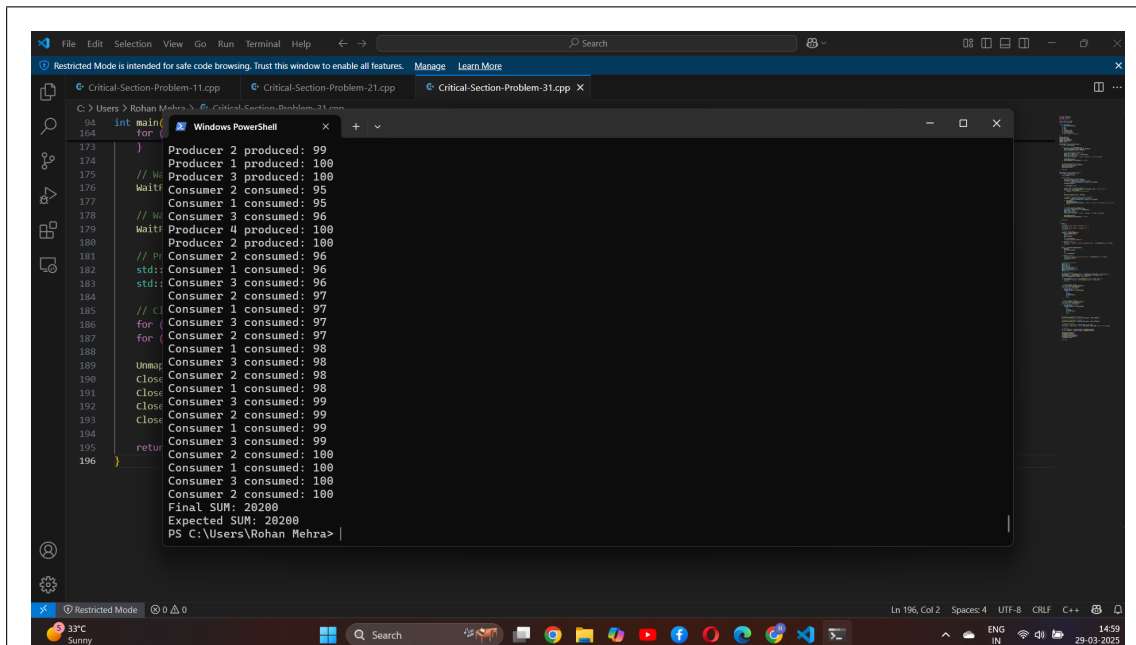
$$\text{SUM} = m \times \frac{100 \times (100 + 1)}{2} = m \times 5050$$

- For example, with  $m = 4$  and  $n = 3$ , the expected SUM is:

$$\text{SUM} = 4 \times 5050 = 20200$$

- The program ensures that no numbers are missed

**Code:** The implementation can be found in the GitHub repository: [GitHub Repository](#).



The screenshot shows a Windows IDE with a C++ file named 'Critical-Section-Problem-31.cpp'. The code is a multi-threaded program using `std::thread` and `std::mutex` to simulate a producer-consumer scenario. It features three producers and three consumers. A `Windows PowerShell` terminal window is open, displaying the program's output. The output shows a sequence of 'Produced' and 'Consumed' messages from different threads, indicating concurrent execution. At the end, it prints 'Final SUM: 20200' and 'Expected SUM: 20200', confirming the correctness of the synchronization. The taskbar at the bottom shows the system clock as 14:59 on 29-03-2023.

```
int main()
{
    // ... (code omitted for brevity) ...
    Producer 2 produced: 99
    Producer 1 produced: 100
    // ... (code omitted for brevity) ...
    Consumer 2 consumed: 95
    Consumer 1 consumed: 95
    // ... (code omitted for brevity) ...
    Consumer 1 consumed: 97
    Consumer 3 consumed: 97
    for (
    for (
    Consumer 1 consumed: 98
    Consumer 3 consumed: 98
    Unmap
    Close
    Consumer 1 consumed: 98
    Consumer 3 consumed: 99
    Close
    Consumer 2 consumed: 99
    Close
    Consumer 1 consumed: 99
    Consumer 3 consumed: 99
    Consumer 2 consumed: 100
    Consumer 1 consumed: 100
    Consumer 3 consumed: 100
    Consumer 2 consumed: 100
    Final SUM: 20200
    Expected SUM: 20200
    PS C:\Users\Rohan Mehra>
```

### Sample Output:

```
Processed element at index 0: 12
Processed element at index 50: 12
Processed element at index 100: 12
...
```

The output demonstrates that:

- Both threads process elements concurrently.
- The threads are canceled after 1 second.
- The main thread confirms successful cancellation.