



Operating Systems

Processes-Part3

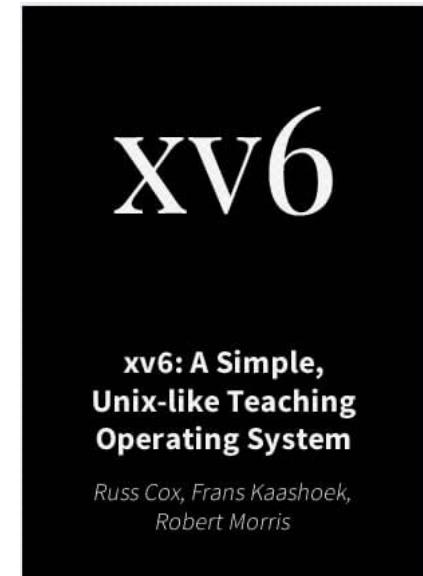
Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Fall 2021

Course logistics

- You have **less than a week** to complete
 - Phase 1 of the project
 - Your first homework
- We will have an extra session on fork
 - The session is handled by TAs
 - It covers both theoretical aspects and how use fork in practice
 - Most probably at coming Thursday



* pid نمایندگی متفاوتان.

* exec تک برآورده ای را در فرزنده ایجاد می کنند. (کل جایگزین سیستم ابزارهای پیش) ← هرجایی در سیستم فرزنده exec صاریح نیست، کل برواب و متغیر ها در... جایگزین سیستم با فرزنده جبرانی.



Process Termination

- Process executes last statement and then asks the operating system to **delete it** using the **exit()** system call.

exit()

- Returns status data from child to parent (via **wait()**)
- Process' resources are deallocated by operating system.

allocate = اینها ممکن نیستند

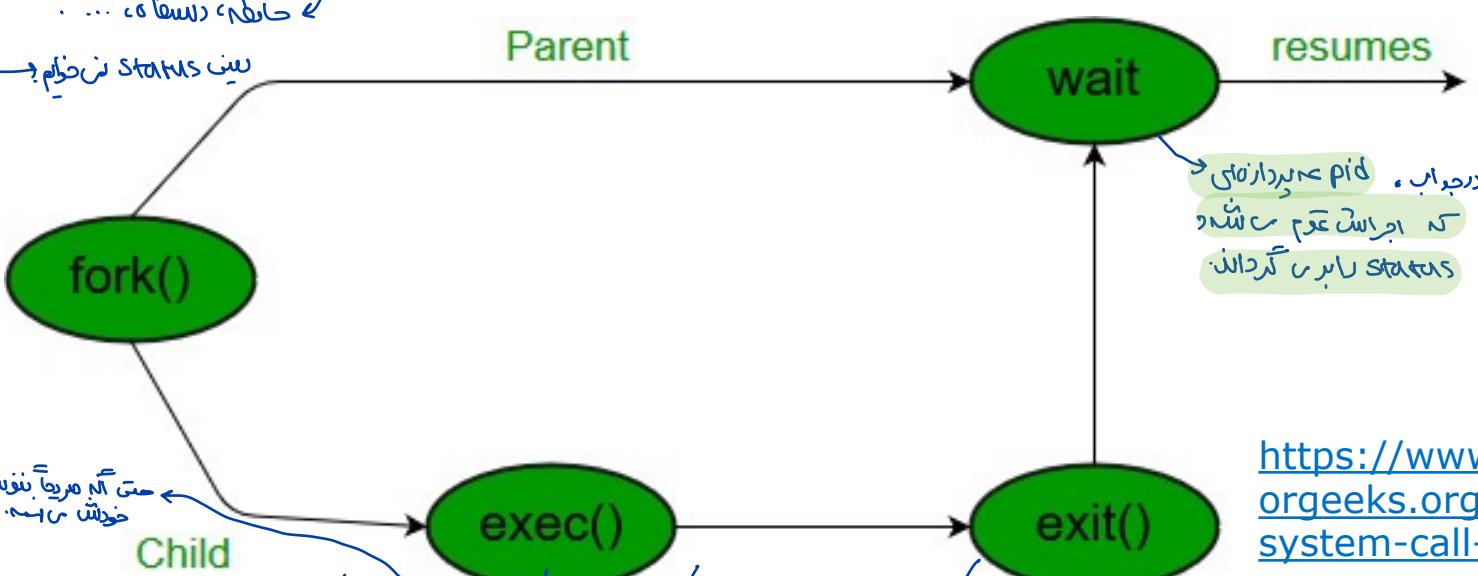
سیستم کردن!

ین باتلاق های را برای دفعه اول
در پروتکل زیر تامین مخاطب می کنند
اجزای فرزند نیستند.

Parent

wait

resumes

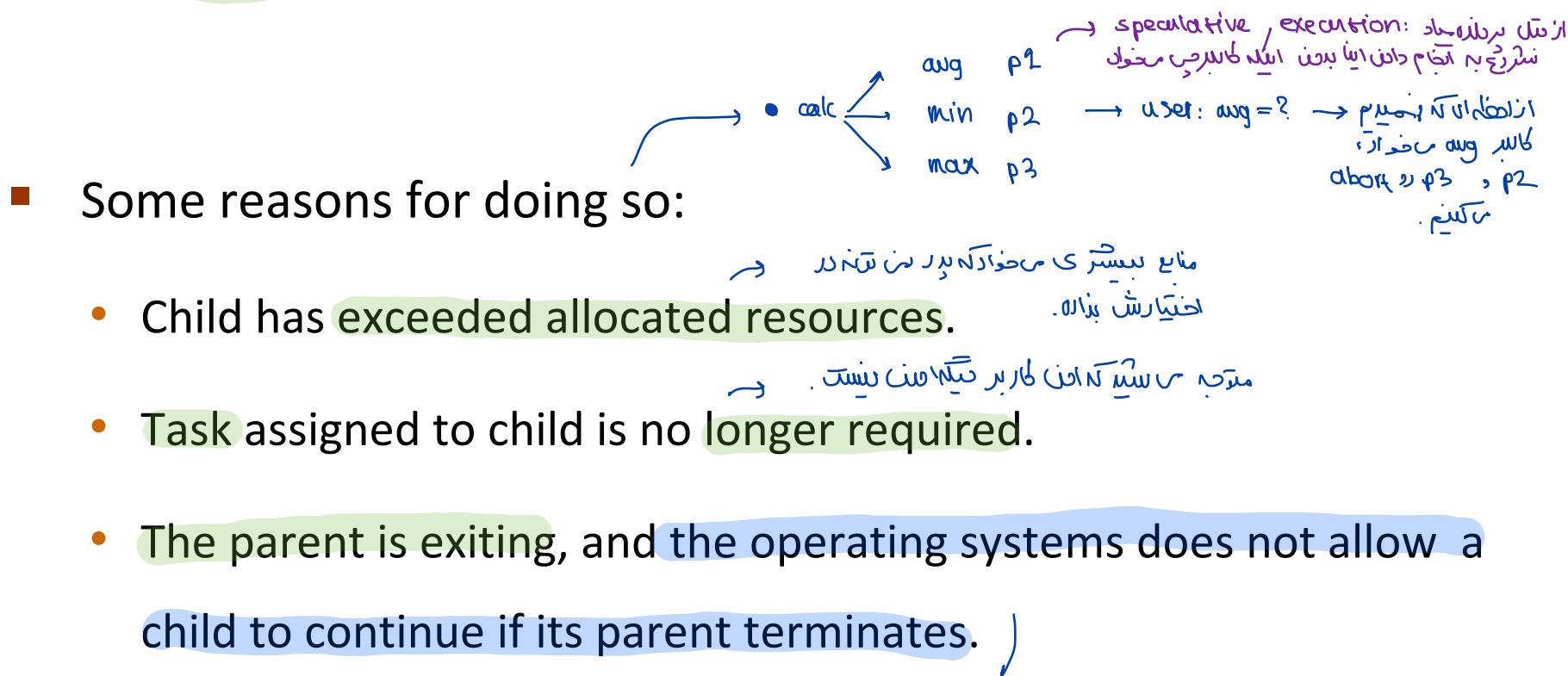


<https://www.geeksforgeeks.org/wait-system-call-c/>



Process Termination (cont.)

- Parent may terminate the execution of children processes using the **abort()** system call.

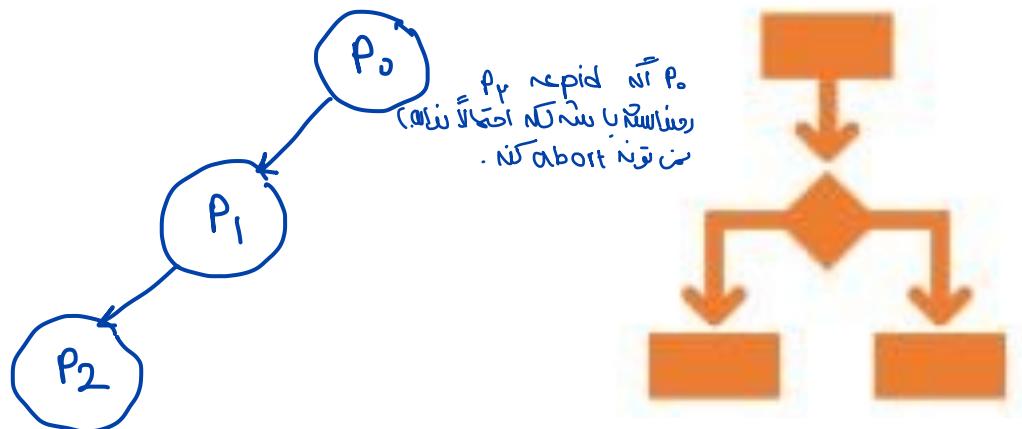


Process Termination (cont.)

- Some OSs do not allow child to **exists** if its parent has terminated.
 - If a process terminates, then all its children must also be terminated.
حالات دستبرد
 - **Cascading termination:** All children, grandchildren, etc., are terminated.
(ین داخل پاره سازی exit تری ارن OS وجودداره.)
 - The termination is initiated by the operating system.



* در این میکن مخفف دعو دلالت بالش. چون هر لر کنی ترجمه شد بالش متذکر شده است.



Process Termination (cont.)

- The parent process may wait for termination of a child process by using the **wait()** system call.
- The call returns status information and the pid of the terminated process.

call by reference

exit, fork, wait, abort, exec

status = Nexit Nfree N僵死 Ngive

for(i=0; i<5; i++)
wait();

هر آنچه از دلخواهی خروجی
نیز تحریم نماید نایاب است

exit نرمند کریم بیاش

pid = wait(&status);

if 0 → success
if -1 → fail

zombie نرمند نایاب کنندی

بعین نور نرمند را ایجاد کنندی

If no parent waiting (did not invoke wait()), process is a zombie.

If parent terminated without invoking wait(), process is an orphan.

بنای باریم برخلاف طایف سیم دی مارس زامبی

سیم دی زامبی

سوالاتی اسکان سوچون ← سوچون ← سوچون

harvard - MIT OS midterm questions: ...

zombie = child's entry still exists in process table although it has called "exit()".

consists of PCBs.

زون نیاز استم ، deallocate ، بالتنی

zombie ← نایاب نایاب نایاب ← نایاب child

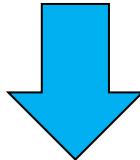
Operating System Concepts--S.A.Javadi

~~over threads~~ Multiprocess Architecture – Browser

- Many web browsers ran as single process (some still do)

(control flow / context switch). If crash happens in one site, it can affect the entire browser.

If one web site causes trouble



Entire browser can hang or crash

Multiprocess Architecture – Chrome Browser (cont.)

- Google Chrome is multiprocess with 3 different types of processes:

- Browser process manages user interface, disk and network I/O.
- Renderer process renders web pages, deals with HTML, Javascript.
presents = ↗

▶ A new renderer created for each website opened

▶ Runs in sandbox restricting disk and network I/O (why?)

- Plug-in process for each type of plug-in.

↳ software component that adds a feature to an existing program.



Each tab represents a separate process.

docke container : sand box مکانیزم از

Inter-Process Communication

- Processes within a system may be **independent** or **cooperating**
امن متوجه حضور عدم ظیفشن.
- Cooperating process can affect or be affected by other processes, including **sharing data**.
- Reasons for cooperating processes:
 - Information sharing
 - Computation speedup
 - Modularity
 - Convenience

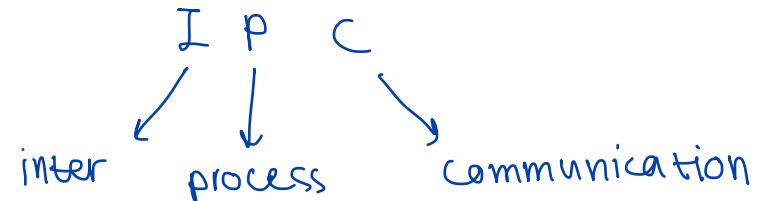
از این سوال تُردی زیادتی امکان عیاد.

چند تابرو نای آن با هم می توانند کار کنند
سیستم بالاتر می تواند راه را درست نهاد.

Inter-Process Communication (Cont.)

پردازه‌های مشارکتی

- Cooperating processes need interprocess communication (IPC)



- Two models of IPC

- Shared memory
- Message passing

► We do not cover this.

Communications Models

خواص رم محدود شده با اسم **حافظه مسیری** معرفه شدند.

و همچنان **shared memory** معرفه شدند.

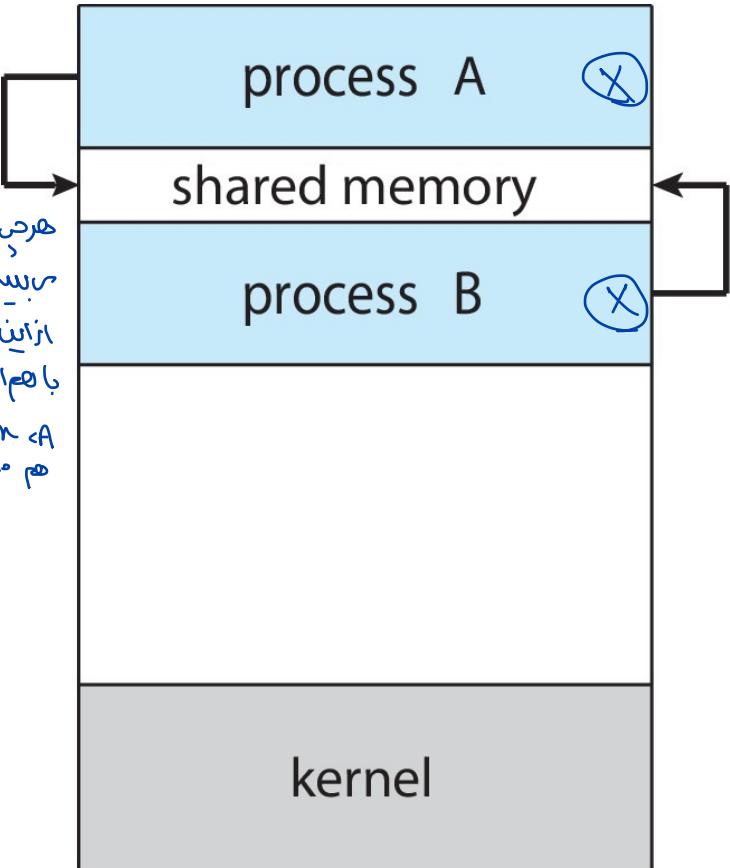
+ این مدلی هم جزوی سریع می باشد؟

و هم

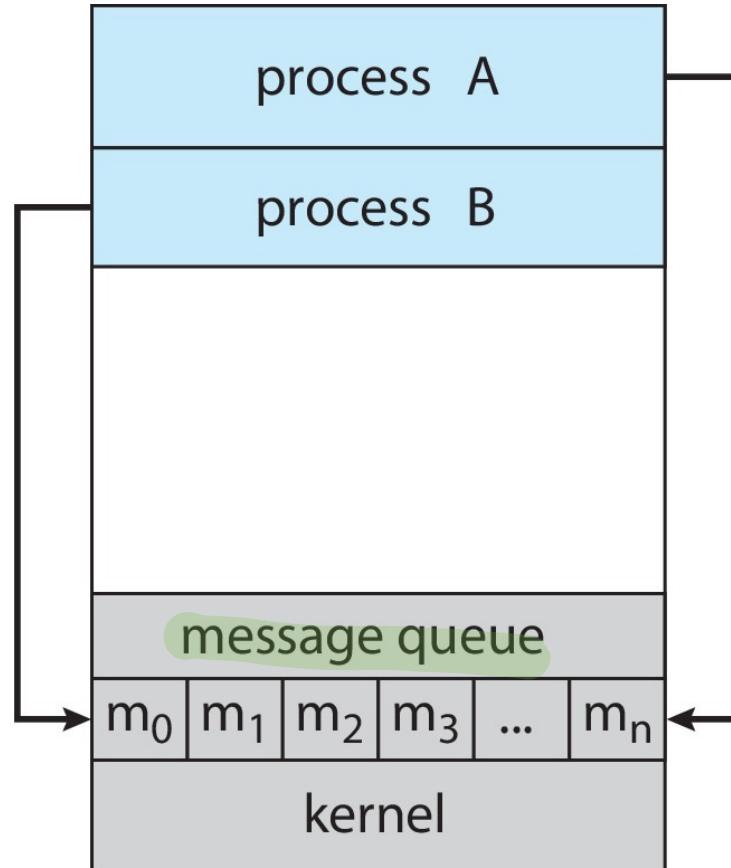
(a) Shared memory.

نه! دسترسی به این پرطه ها باید باشد.

(b) Message passing.



(a)



(b)

control سند یافته، ملکت برداشتن را ناید لذت.



Producer-Consumer Problem

- Paradigm for cooperating processes:
 - **Producer** process produces information that is consumed by a **consumer** process.

- Two variations:
 - **Unbounded-buffer** places no practical limit on the size of the buffer:
 - ▶ Producer never waits
 - ▶ Consumer waits if there is no buffer to consumer
 - **Bounded-buffer** assumes that there is a fixed buffer size
 - ▶ Producer must wait if all buffers are full
 - ▶ Consumer waits if there is no buffer to consume



IPC – Shared Memory

- An area of memory shared among the processes that wish to communicate.
- The communication is ***under the control of the users*** processes ***not the operating system***.
- Major issues is to provide mechanism that will allow the user processes ***to synchronize their actions*** when they access shared memory.
- Synchronization is discussed in great details in Chapters 6 & 7.

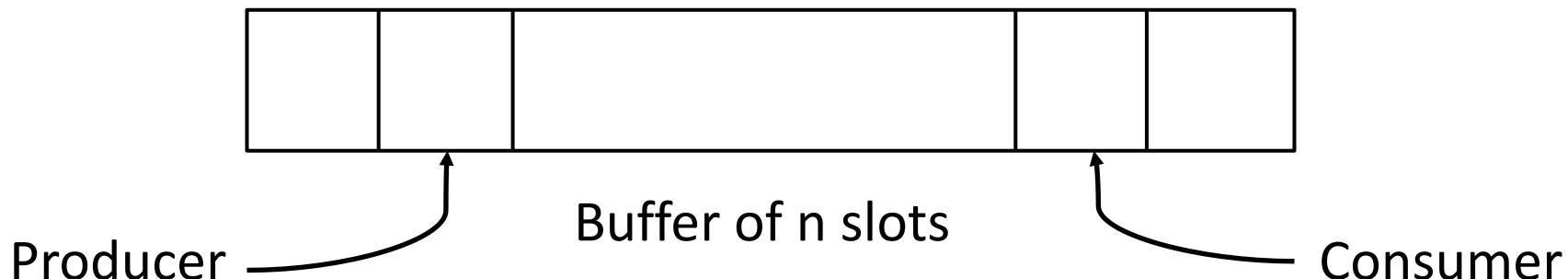


Bounded-Buffer – Shared-Memory Solution

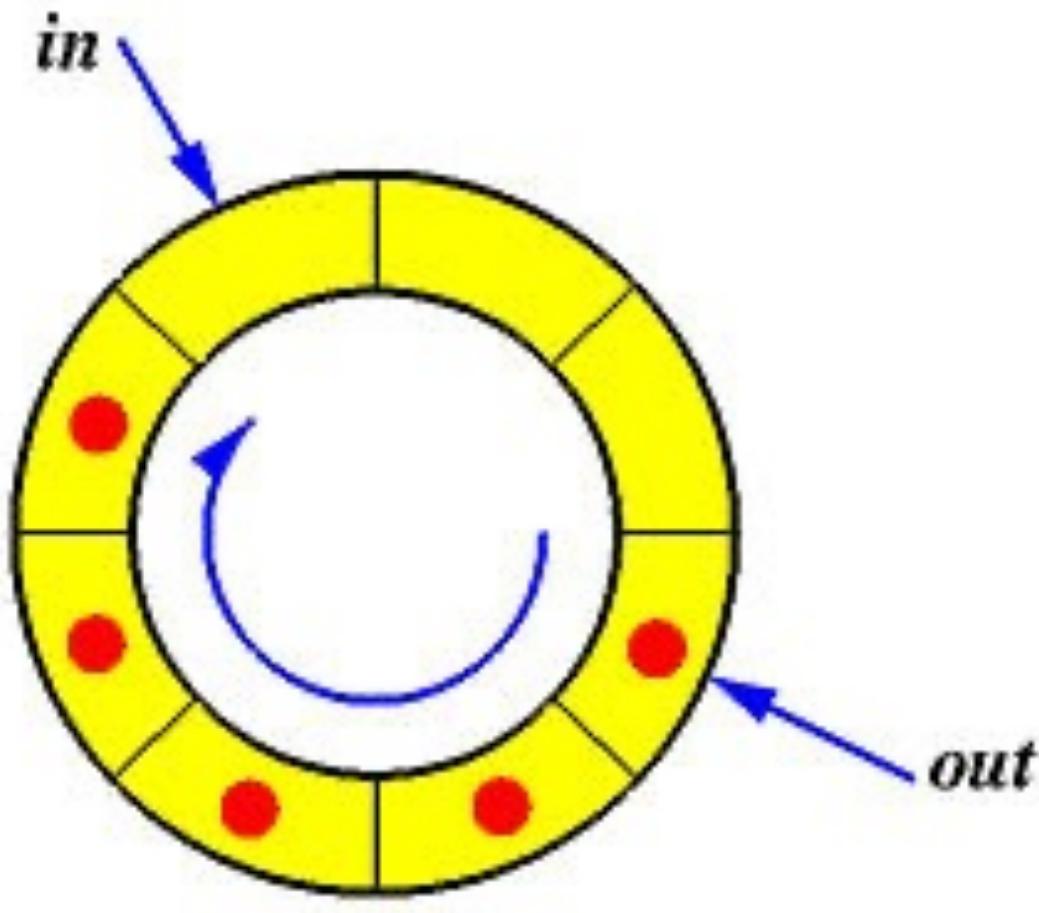
- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
    . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Solution is correct but can only use **BUFFER_SIZE - 1** elements.



Circular Bounded-Buffer



Source: <https://pages.mtu.edu/~shene/NSF-3/e-Book/SEMA/TM-example-buffer.html>

Producer Process – Shared Memory

```
item next_produced;

while (true) {
    /* produce an item in next produced */
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = next_produced;
    in = (in + 1) % BUFFER_SIZE;
}
```



Consumer Process – Shared Memory

```
item next_consumed;

while (true) {
    while (in == out)
        ; /* do nothing */
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;

    /* consume the item in next consumed */
}
```



What about Filling all the Buffers?

- Suppose that we wanted to provide a solution to the consumer-producer problem that **fills all the buffers**.
- How can we do it?



What about Filling all the Buffers? (ont.)

- We can do so by having *an integer counter* that keeps track of the number of full buffers.
- Initially, counter is set to 0.
- The integer counter is incremented by the producer after it produces a new buffer.
- The integer counter is decremented by the consumer after it consumes a buffer.



Producer

```
while (true) {  
    /* produce an item in next_produced */  
  
    while (counter == BUFFER_SIZE)  
        ; /* do nothing */  
    buffer[in] = next_produced;  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;  
}
```



Consumer

```
while (true) {  
    while (counter == 0)  
        ; /* do nothing */  
    next_consumed = buffer[out];  
out = (out + 1) % BUFFER_SIZE;  
    counter--;  
    /* consume the item in next  
consumed */  
}
```

Race Condition

- **counter++** could be implemented as

```
register1 = counter  
register1 = register1 + 1  
counter = register1
```

- **counter--** could be implemented as

```
register2 = counter  
register2 = register2 - 1  
counter = register2
```



Race Condition (cont.)

- Consider this execution interleaving with “count = 5” initially:

S0: producer execute **register1 = counter** {register1 = 5}

S1: producer execute **register1 = register1 + 1** {register1 = 6}

S2: consumer execute **register2 = counter** {register2 = 5}

S3: consumer execute **register2 = register2 - 1** {register2 = 4}

S4: producer execute **counter = register1** {counter = 6 }

S5: consumer execute **counter = register2** {counter = 4 }



Race Condition (cont.)

Question – why was there no race condition in the first solution
(where at most $N - 1$) buffers can be filled?

More in Chapter 6.

