OPERATING SYSTEMS: PROCESS COMMUNICATION AND SYNCHRONIZATION

Concurrent Processes and problems in communication and synchronization

Contents

- **Concurrency.**
- Race conditions.
- Mutual exclusion and critical section.
- Semaphores.
- Producer-Consumer problem.
- Readers-Writers problem.

Concurrent process

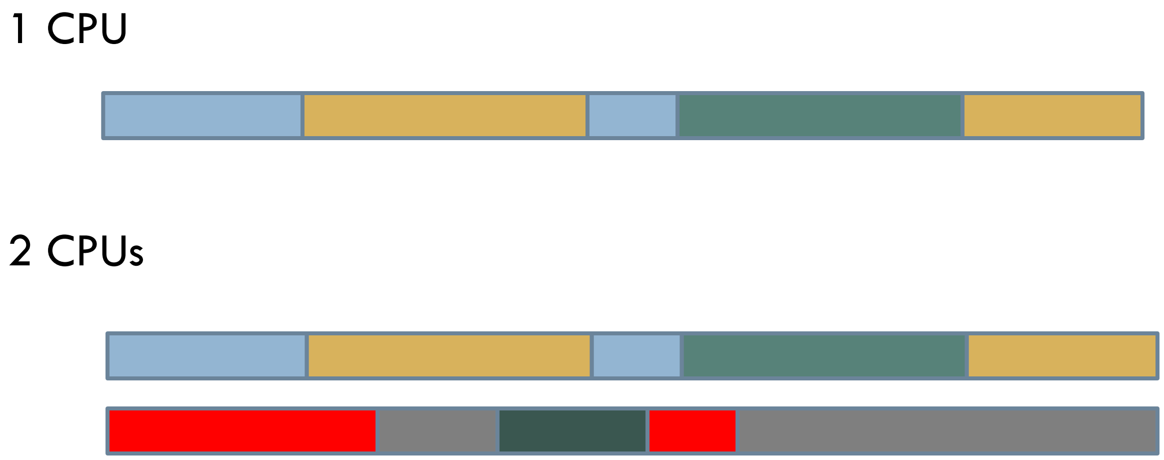
- Two processes are **concurrent** when they are executed in a way that their **execution intervals overlap**.

There is concurrency

There is NO concurrency

Kinds of concurrency

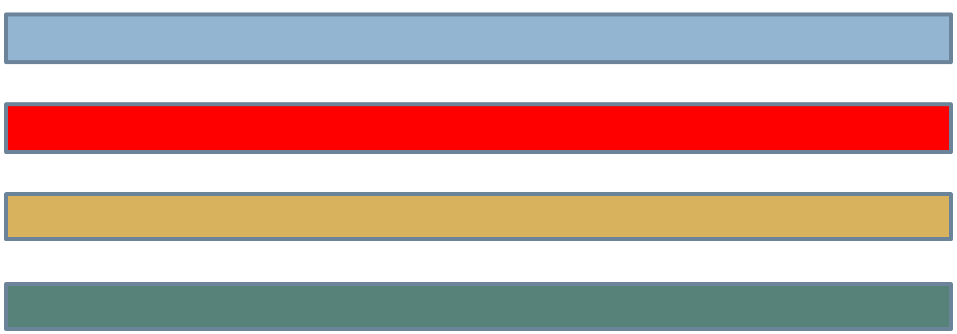
- **Apparent concurrency**: More processes than processors
  - Processes are multiplexed in time.
  - Pseudoparallelism.



Kinds of concurrency

- **Real concurrency**: Each process runs in a processor.
  - A **parallel execution** happens.
  - Also known as **real parallelism**..

4 CPUs



Concurrent programming models

- **Multiprogramming** with single processor.
  - OS in charge of dividing time among processes (appropiative/non-appropriative scheduling).
- **Multiprocessor**.
  - Combines real parallelism and pseudoparallelism.
    - Usually more processes than CPUs.
- **Distributed system**.
  - Several computers connected through a network.

Advantages of concurrent execution

- **Eases programming.**
  - Several tasks can be structured in separate processes.
  - **Web server**: A process in charge of handling each request.
- **Accelerates computation execution.**
  - Division of computations in processes run in parallel.
  - **Examples**: Simulations, Image processing, energy market, financial computations.
- **Improves application interactivity.**
  - Ability to separate processing tasks from user attention tasks.
  - **Example**: Print and edit.
- **Improve CPU utilization.**
  - I/O phases from one application are used for processing from other applications.

Kinds of concurrent processes

- Independent.
  - Processes run concurrently but without interaction.
    - No need for communication.
    - No need for synchronization.
    - Example: Two shells for two users run in two different terminals.

- Cooperating.
  - Processes run concurrently with some interaction.
    - May communication each other.
    - May be synchronized.
    - Example: Transaction server organized in receiver process and request handling processes.

Interactions among processes

- **Access to shared resources**:
  - Processes sharing a resource.
  - Processes competing for a resource.
  - **Example**: Request server with different processes writing to a log file.

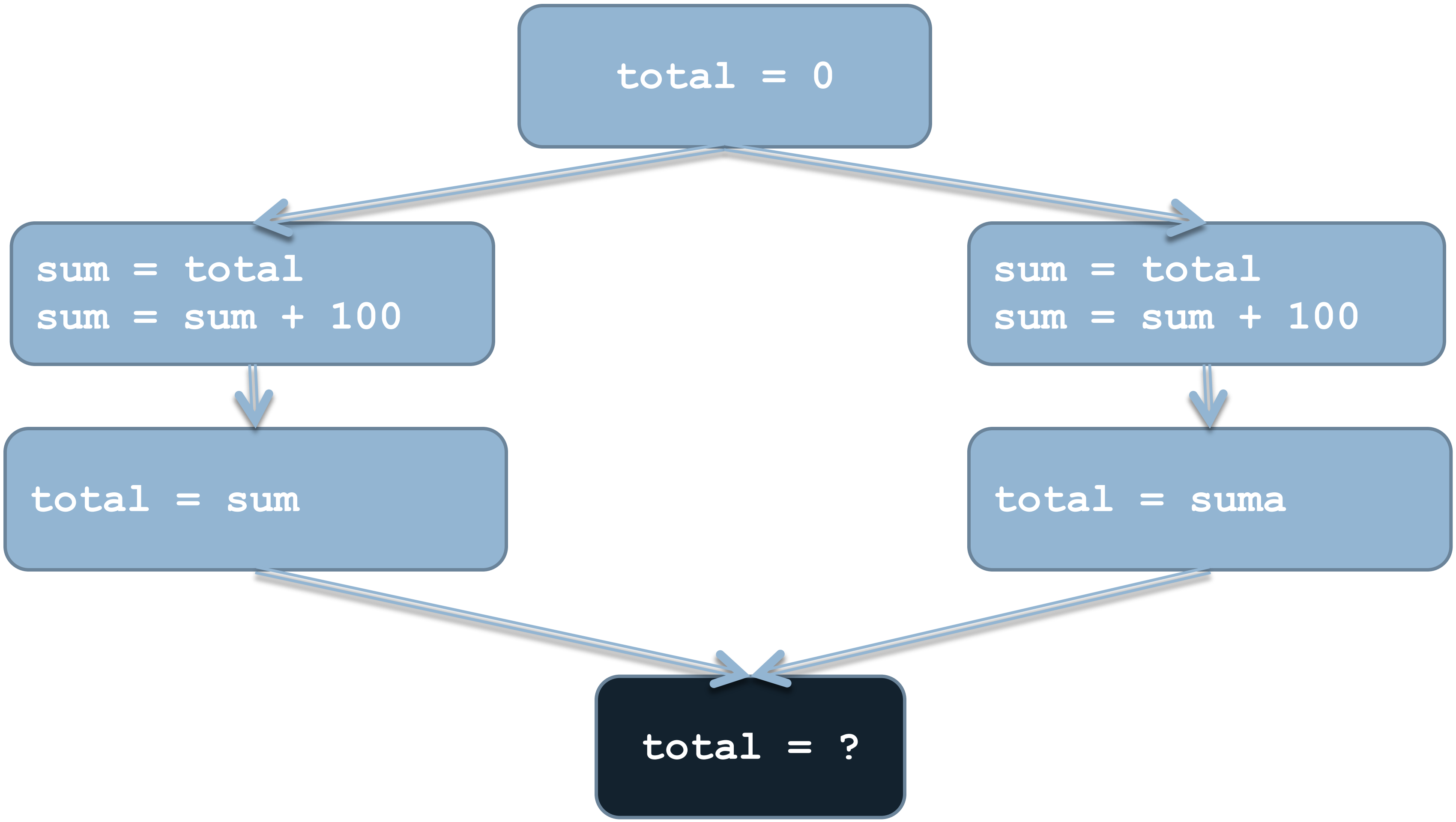
- **Communication**:
  - Processes exchanging information.
  - **Example**: Requests receiver must pass information to a request handling process.

- **Synchronization**:
  - A process must wait for an event in another process.
  - **Example**: A user interface process must wait until all computation process have finished.

Contents

- Concurrency.
- **Race conditions.**
- Mutual exclusion and critical section.
- Semaphores.
- Producer-Consumer problem.
- Readers-Writers problem.

Race condition



Race condition


```
#include <stdio.h>
#include <pthread.h>

#define NUMTH 10
int total = 0;

void add() {
  int i,n;
  int sum = total;
  sum = sum + 100;
  n=rand()%5;
  for (i=0;i<n;i++)
   {printf(".");}
  total = sum;
}
```

```
int main() {
  pthread_t th[NUMTH];
  int i;
  for (i=0;i<NUMTH;i++) {
    pthread_create(&th[i],
      NULL,(void*)add, NULL);
  }

  for (i=0;i<NUMTH;i++) {
    pthread_join(th[i], NULL);
  }

  printf("Result=%d\n",
    total);
}
```

Result?

Result


```
[jdaniel@tucan ~]$ ./test2
................Suma=200
[jdaniel@tucan ~]$ ./test2
................Suma=600
[jdaniel@tucan ~]$ ./test2
................Suma=500
[jdaniel@tucan ~]$ ./test2
................Suma=300
[jdaniel@tucan ~]$ ./test2
................Suma=600
[jdaniel@tucan ~]$ ./test2
................Suma=600
[jdaniel@tucan ~]$ ./test2
................Suma=500
[jdaniel@tucan ~]$ ./test2
................Suma=600
[jdaniel@tucan ~]$ ./test2
................Suma=600
[jdaniel@tucan ~]$ ./test2
................Suma=600
[jdaniel@tucan ~]$ ./test2
................Suma=500
```

- Each run gives a different result.
- Never gets the correct result.
- What's happening?

Possible sequences

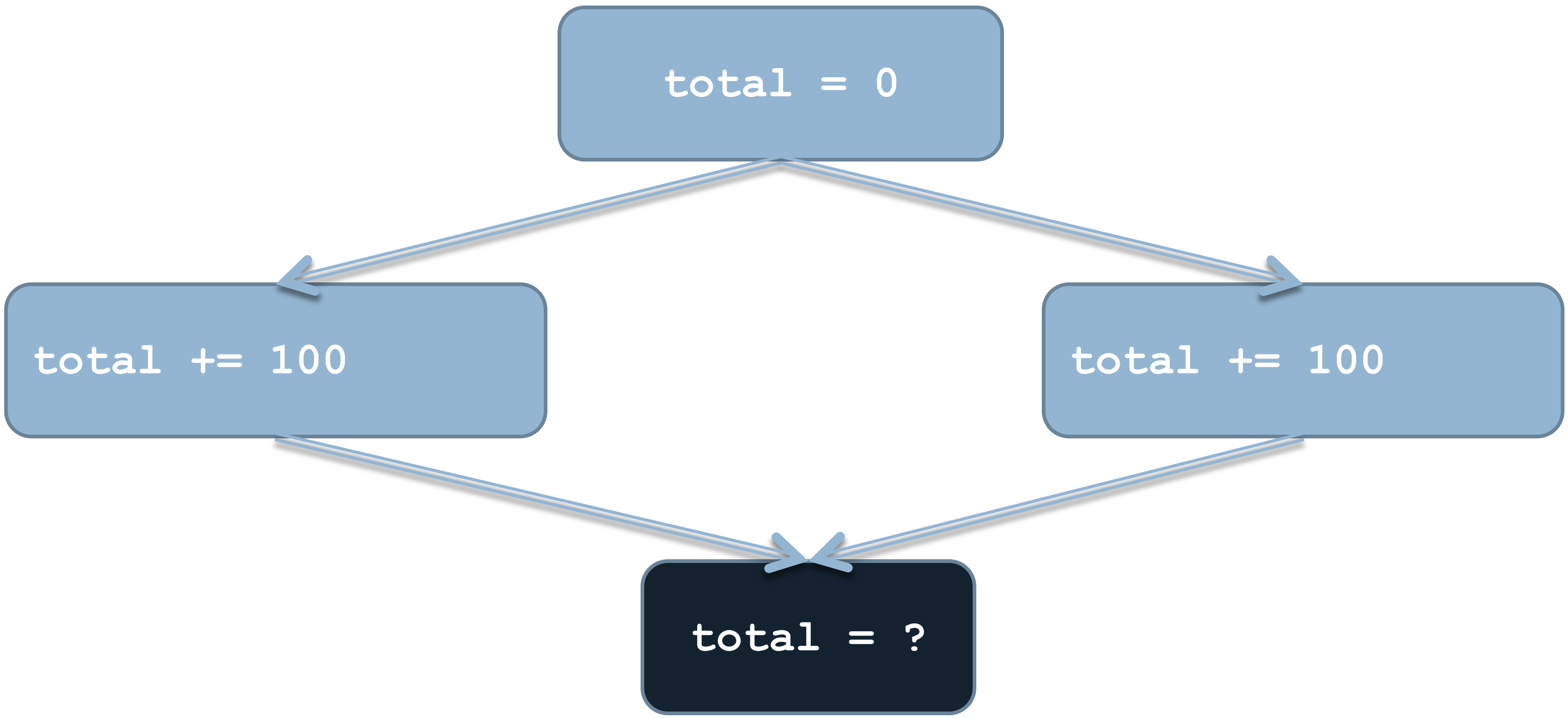

```
total = 0
sum = total
sum = sum + 100
total = sum
sum = total
sum = sum + 100
total = sum
```

200

```
total = 0
sum = total
sum = sum + 100
sum = total
sum = sum + 100
total = sum
total = sum
```

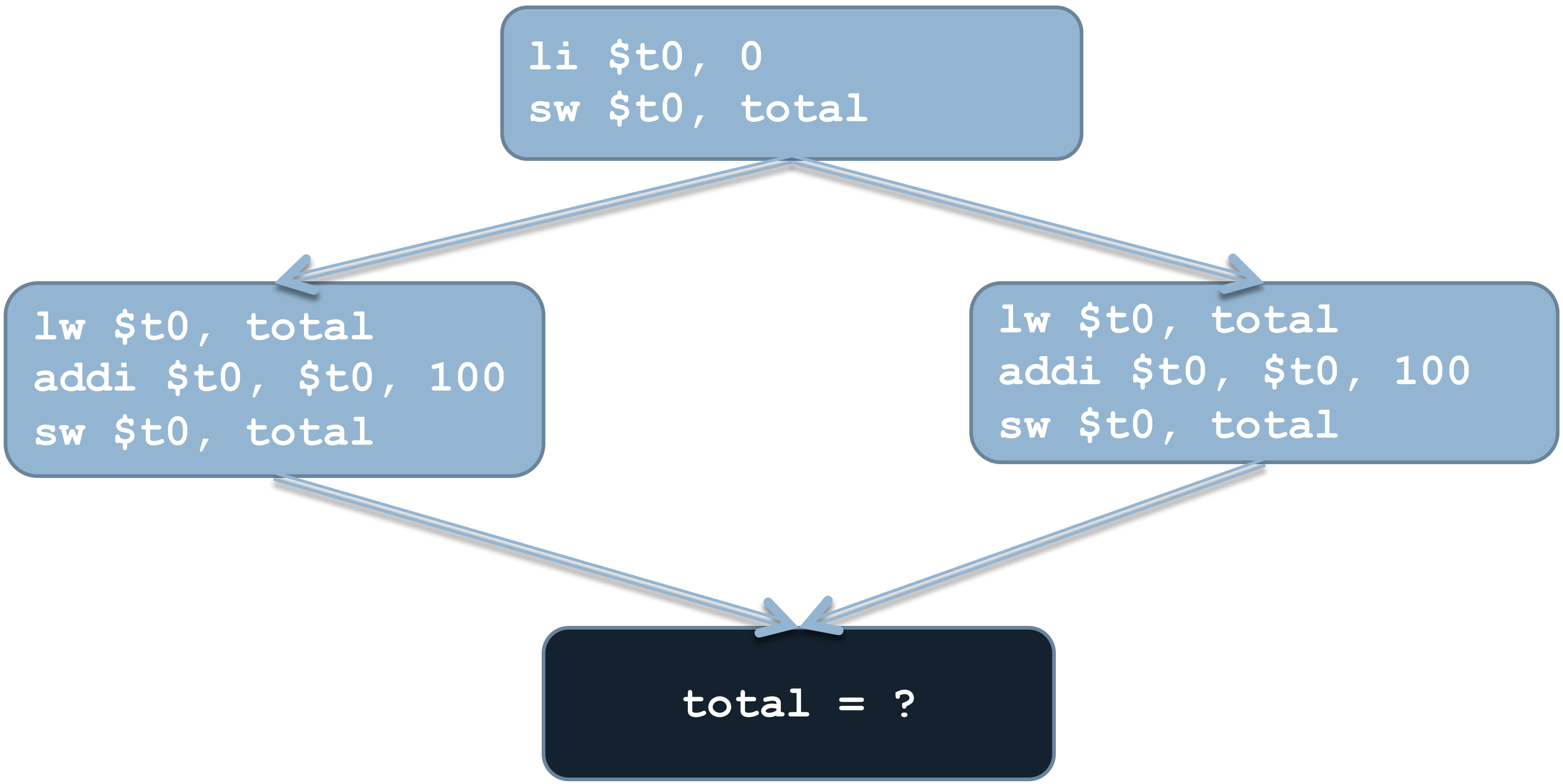
100

Let's try again



Is a race condition possible?

Machine instructions



Can this happen in a multiprocessor?

Race conditions

- Functioning of a process and its results must be independent from its relative execution speed with respect to other processes.
  - Necessary to guarantee that order of execution does not affect result.

- **Solution**: Achieve that a set of instructions can be executed atomically.

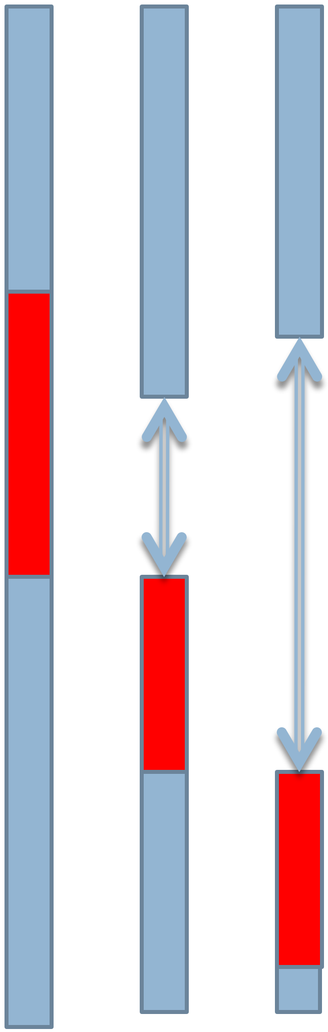
Mutual exclusion

Contents

- Concurrency.
- Race conditions.
- **Mutual exclusion and critical section.**
- Semaphores.
- Producer-Consumer problem.
- Readers-Writers problem.

Mutual exclusion

- **Critical section**: Code segment manipulating a resource that must be atomically executed.
- A resource is associated to a management mechanism for mutual exclusion.
- Only a process may be at the same time in the critical section of a resource.



Problems with critical sections

- Deadlocks.
  - Produced with mutual exclusion for multiple resources.
    - Process P1 enters critical section for resource A.
    - Process P2 enters critical section for resource B.
    - Process P1 requests to enter into critical section for resource B.
      - Keeps waiting P2 to exit from critical section for resource B.
    - Process P2 requests to enter into critical section for resource A.
      - Keeps waiting P1 to exit from critical section for resource A.

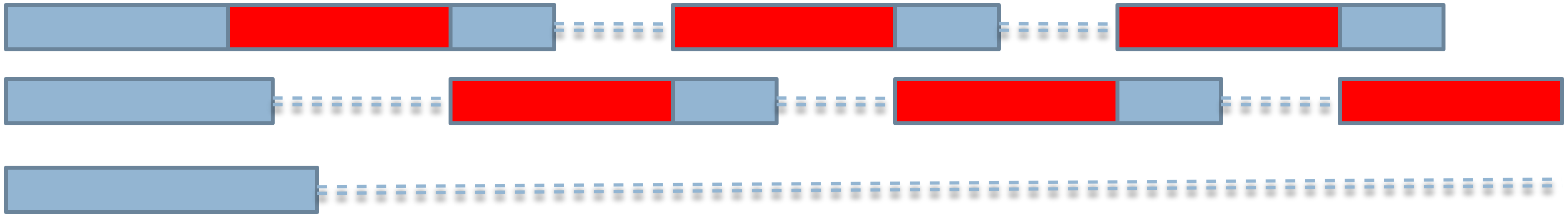
None of them can make progress

Problems with critical sections

- Starvation.
  - A process keeps indefinitely blocked waiting to enter a critical section.
    - Process P1 enters critical section for resource A.
    - Process P2 requests to enter into critical section for resource A.
      - Keeps waiting for P1 to exit from critical section for resource A.
    - Process P3 requests to enter into critical section for resource A
      - Keeps waiting for P1 to exit from critical section for resource A.
      - P2 and P3 are blocked.
    - Process P1 leaves critical section for resource A.
    - Process P2 enters critical section for resource A.
      - P3 is blocked.
    - Process P1 requests to enter into critical section for resource A.
      - Keeps waiting for P2 to exit from critical section for resource A.
      - P2 and P3 are blocked.
    - Process P2 leaves critical section for resource A.
      - P3 is blocked
    - Process P1 enters into critical section for resource A.
      - P3 is blocked.
    - …

Process P3 never gets to enter into critical section for A.

Starvation



Process P3 never gets to enter into critical section for A.

Conditions for mutual exclusion

- Only one process is allowed to be at the same time into the critical section of a resource.
- It must not be possible for a process that requests to enter into a critical section to be postponed indefinitely.
- When no process is in a critical section, any process requesting to enter that critical section will do so without delay.
- No assumptions about relative speed of processes or about the number of processors must be made.
- A process stays in its critical section for a finite amount of time.

Critical section: Synchronization mechanism

- Any mechanism solving the problem of the critical section must provide synchronization between processes.
  - Each process must request permission to enter into the critical section.
  - Each process must signal when it is exiting the critical section.

```
Non-critical code
…
<Entry to critical section>
Critical section code
<Exit from critical section>
…
Non-critical code
```

Implementation alternatives

- Disable interrupts.
  - Process would not be interrupted.
  - Only valid in mono-processor systems.

- Machine instructions.
  - Test and set or swap.
  - Implies busy waiting.
  - Starvation and deadlock are possible.

- Other alternative: Operating System support.

Contents

Semaphores (Dijkstra)

- Process synchronization through a signaling mechanism.
  - Semaphore.

- A semaphore may be seen as an integer variable with three associated operations.
  - Initialize to a non-negative value.
  - **semWait**: Decrements the counter
    - If s<0 → Process blocks.
  - **semSignal**: Increments value of counter.
    - If s<=0 → Unblocks process.

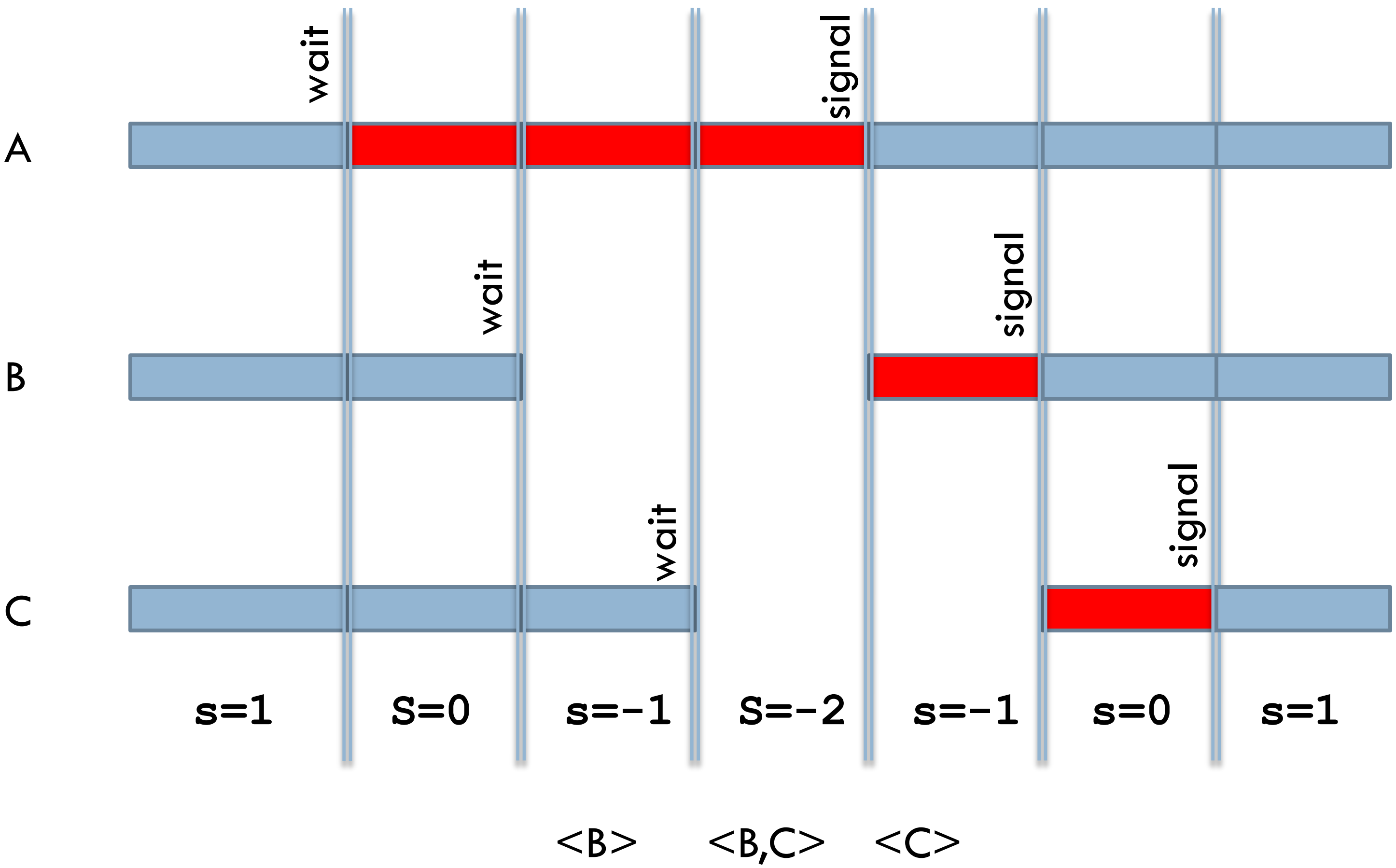
Atomic Operations

Critical sections and semaphores

- A semaphore associated to the critical section of a resource.
- Semaphore initialized to 1.
- Entry to critical section: **semWait**.
- Exit from critical section: **semSignal**.

```
Non-critical code
…
semWait(s);
Critical section code
semSignal(s);
…
Non-critical code
```

Critical sections and semaphores

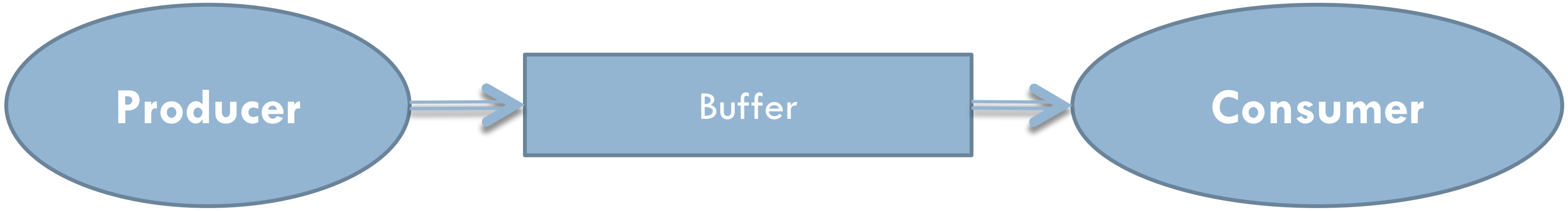


Contents

- Concurrency.
- Race conditions.
- Mutual exclusion and critical section.
- Semaphores.
- **Producer-Consumer problem.**
- Readers-Writers problem.

Producer-Consumer problem

- A process **produces** information elements.
- A process **consumes** information elements.
- There is an **intermediate** storage space (buffer).



Infinite buffer

Producer

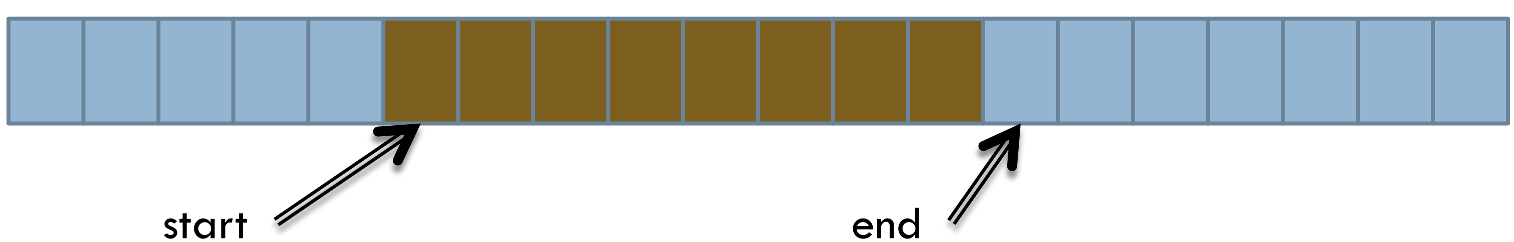
```
for (;;) {
   x= produce();
   v[end] = x;
   end++;
}
```

Synchronization needed

Consumer

```
for (;;) {
   while (start==end)
      {}
   y=v[start];
   start++;
   process(y);
}
```

Busy waiting



Infinite buffer

`semaphore s=1`

Producer	Consumer

```
for (;;) {
   x= produce();
   semWait(s);
   v[end] = x;
   end++;
   semSignal(s);
}
```

```
for (;;) {
   while (start==end)
      {}
   semWait(s);
   y=v[start];
   start++;
   semSignal(s);
   process(y);
}
```

Busy waiting

Infinite buffer


```
semaphore s=1; semaphore n=0;
```

Producer	Consumer

```
for (;;) {
   x= produce();
   semWait(s);
   v[end] = x;
   end++;
   semSignal(s);
   semSignal(n)
}
```

```
int m;
for (;;) {
   semWait(n);
   semWait(s);
   y=v[start];
   start++;
   semSignal(s);
   consume(y);
}
```

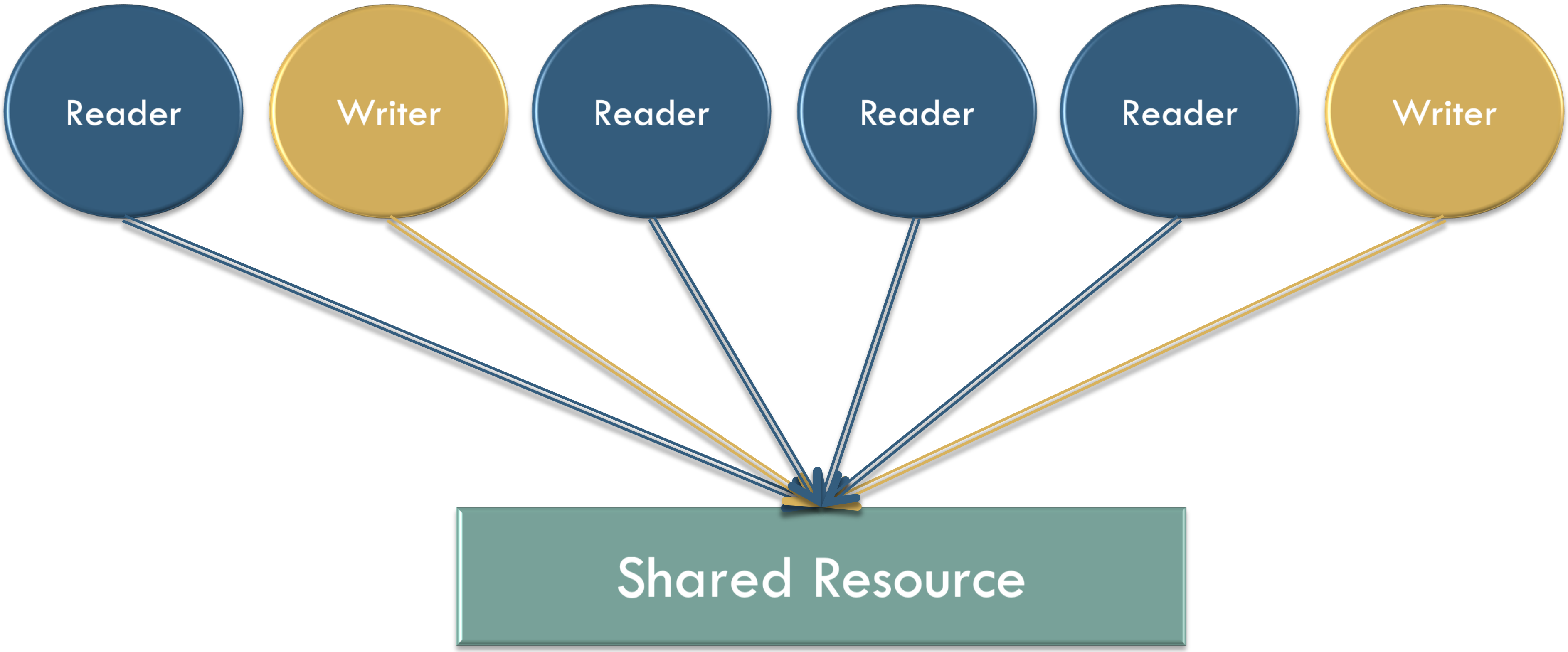
Contents

Readers-Writers problem

- Problem related to a shared area of storage.
  - Multiple processes reading information.
  - Multiple processes writing information.

- **Conditions**:
  - Any number of readers may read from data area concurrently.
  - Only a writer may modify information at once.
  - During a write no reader may perform a request.

Readers-Writers problem



Difference with other problems

- **Mutual exclusion**:
  - In case of mutual exclusion only a process would be allowed to access information.
  - No concurrency among readers.

- **Producer consumer**:
  - In producer/consumer both processes modify the shared memory area.

- **Goals of additional restrictions**:
  - Provide a more efficient solution.
  - **Reduce contention.**

Management alternatives

- **Readers have priority.**
  - If there is some reader in the critical section other readers may enter.
  - A writer may only enter into the critical section if there is no other process already in the critical section.
  - Problem: Starvation for writers.

- **Writers have priority.**
  - When a writer wants to enter into the critical section, no new reader as allowed to enter into the critical section.

Readers have priority


```
int nreaders; semaphore readers=1; semapore writers=1;
```

Reader	Writer

Reader

```
for(;;) {
  semWait(readers);
  nreaders++;
  if (nreaders==1)
    semWait(writers);
  semSignal(readers);

  perform_read();

  semWait(readers);
  nreaders--;
  if (nreaders==0)
    semSignal(writers);
  semSignal(readers);
}
```

Writer

```
for(;;) {
  semWait(writers);
  perform_write();
  semSignal(writers);
}
```

Task: Try to design a solution for writers have priority

OPERATING SYSTEMS: PROCESS COMMUNICATION AND SYNCHRONIZATION

Concurrent Processes and problems in communication and syncrhonization