Sistem Operasi: Proses

Septian Cahyadi

Overview

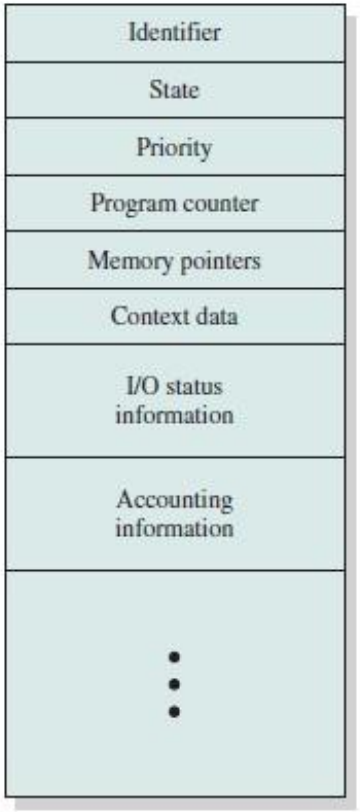
- Event dan Status pada Proses
- Manajemen Proses
- Thread
- Inter Process Communication

Program vs Proses

- Proses adalah program yang sedang dieksekusi
- Untuk bisa dieksekusi berarti harus diload ke memory:
  - Kode program
  - Data
  - Program Counter
  - Stack
  - Nilai-nilai register
- Suatu program yang sama bisa dieksekusi bersamaan menjadi beberapa proses

Process Control Block

- Struktur data yang menyimpan informasi tentang proses
- Karena multiprogramming, setiap proses akan memiliki Process Control Block



Penciptaan Proses

- 4 event yang memicu penciptaan proses:
  - Inisialisasi sistem (booting)
  - Eksekusi system call untuk menciptakan proses (spawning)
  - Permintaan dari user untuk menciptakan proses
  - Awal dari batch job
- Untuk keempat event ini ada satu proses yang akan memanggil system call untuk menciptakan proses baru
- Setiap proses yang diciptakan akan mendapat address space sendiri

Penghentian Proses

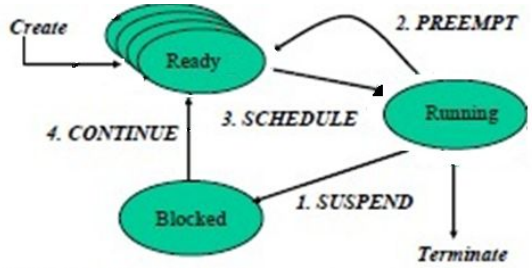
- 4 event yang memicu penghentian proses:
  - Keluar secara normal
  - Keluar karena error, diatur sendiri oleh proses
  - Fatal error, dipaksa oleh sistem
  - Dibunuh proses lainnya, yang memiliki wewenang untuk itu

Hirarki Proses

- Dikenal di Unix, namun tidak di Windows
- Di Unix system call **fork** akan dipanggil untuk membuat proses anak
- Suatu proses dan semua leluhurnya akan membentuk hirarki proses
- Di Unix proses **init** akan dipanggil pertama kali dan menjadi root dari hirarki proses ini

Status Proses

- Setiap proses bisa berada pada salah satu dari 3 status: ready, running, blocked
- Ada 1 proses khusus yang disebut scheduler yang bertugas menjadwalkan proses mana yang bisa running



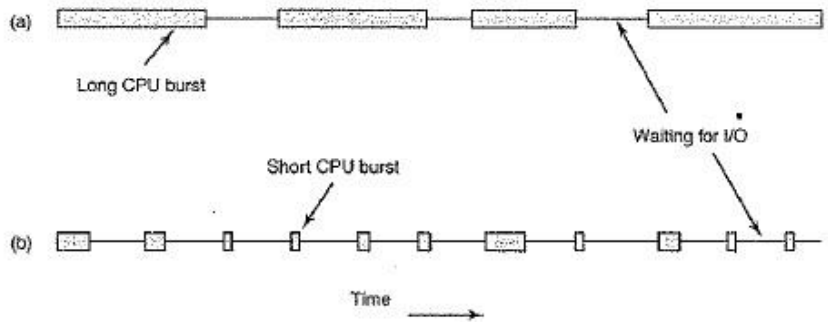
1. Process blocks for I/O
2. Force running process to release the CPU
3. Scheduler picks a ready process to run
4. I/O becomes available

Manajemen Proses

- Status setiap proses dicatat dalam sebuah tabel proses
- Tabel proses adalah struktur data di mana setiap entrinya berupa PCB
- Context switching adalah istilah untuk menyatakan saat scheduler memilih proses lain untuk running

Perilaku Proses & Utilisasi CPU

- Suatu proses tidak terus-menerus memakai CPU



- Dua jenis proses: compute-bound dan I/O-bound
- Dengan multiprogramming utilisasi CPU bisa ditingkatkan
- Jika p adalah proporsi waktu proses menunggu I/O dan ada n proses di memori maka utilisasi CPU = 1 - pn

Contoh Utilisasi CPU

- Jika RAM 512 MB dan OS butuh 128 MB
- Jika tiap proses rata-rata butuh 128 MB memori dan 80% waktu menunggu I/O
- Berarti ada maks 3 proses di memori secara bersamaan
- Utilisasi CPU = 1 – 0.83 = 48.8%
- Jika RAM ditambah 512 MB maka maks ada 7 proses
- Utilisasi CPU = 1 – 0.87 = 79%
- Jika ditambah lagi 512 MB maka maks ada 11 proses
- Utilisasi CPU = 1 – 0.811 = 91%

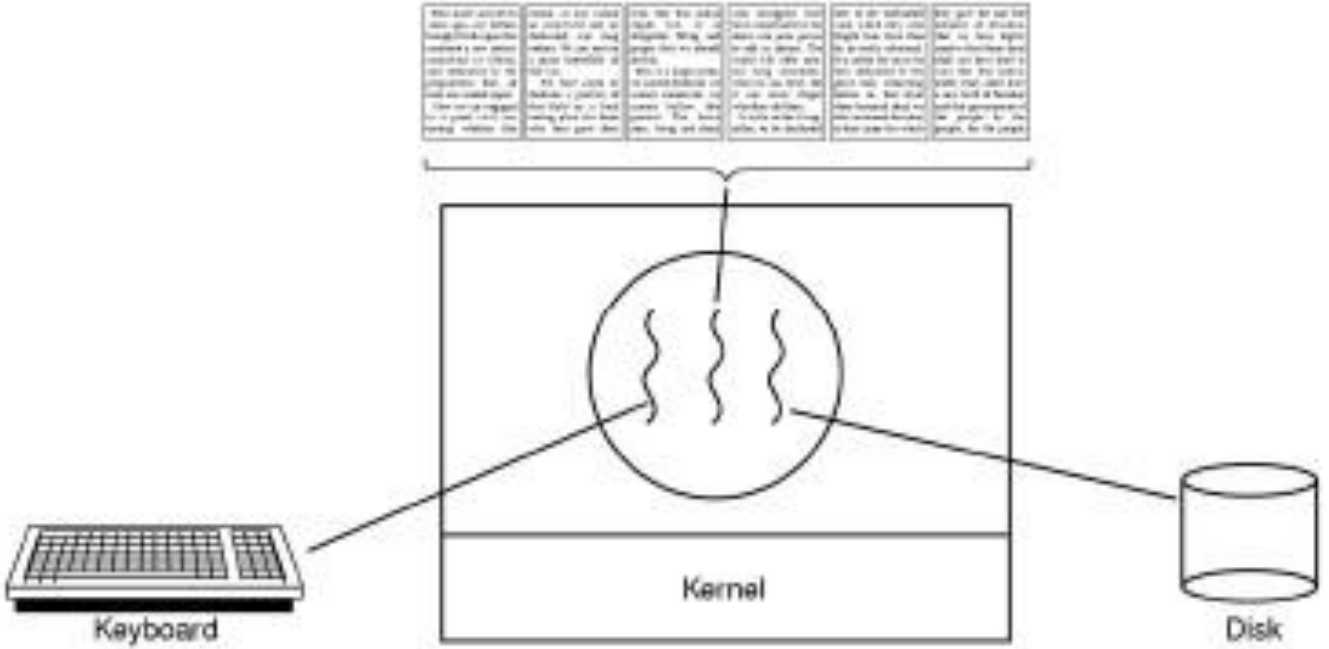
Thread

- Mirip proses namun sharing address space
- Proses di dalam proses
- Mengapa perlu thread? Mengapa tidak menggunakan proses saja?

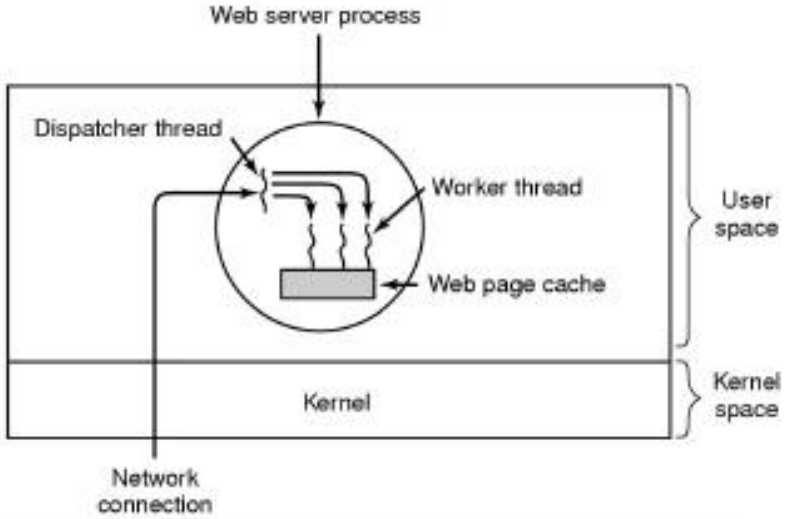
Proses vs. Thread

- Alasan menggunakan thread:
  - Dalam suatu aplikasi berjalan beberapa hal secara paralel, namun perlu sharing data
  - Thread lebih mudah dan cepat untuk diciptakan/dihentikan
- Contoh: word processor digunakan untuk mengedit suatu dokumen besar
- Jika suatu kalimat pada hal. 1 dihapus, lalu harus pindah ke hal. 600, maka akan makan waktu lama
- Akan terasa lebih cepat jika dibagi menjadi 2 thread: reformat dan UI

Word Processor



Word Processor




```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

Manajemen Thread

- Konsep multithreading
- Perlu penanganan karena banyak masalah bisa terjadi pada sharing address space
- Thread table

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

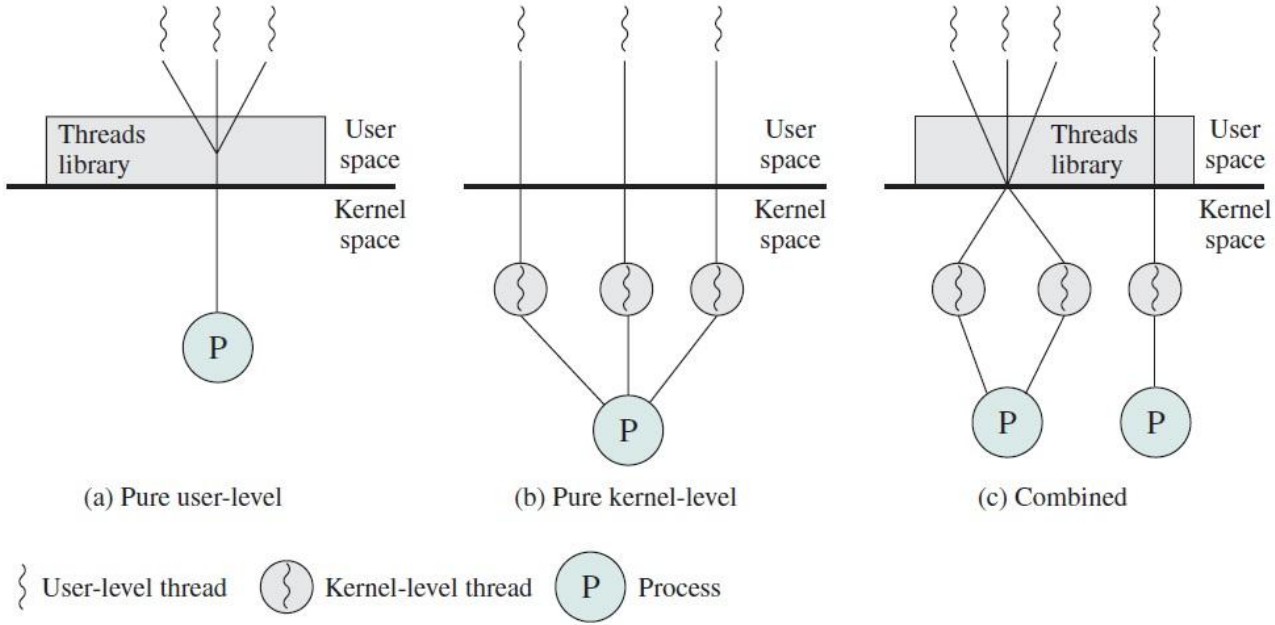
Proses vs. Thread

- Implementasi di user space, dengan library (User Level Threads/ULT).

  Bagi kernel tiap proses hanya punya 1 thread
  - Bisa diimplementasikan bahkan pada SO yang tidak mengenal thread
  - Lebih cepat
  - Scheduler bisa diimplementasikan sendiri
- Implementasi di kernel space (Kernel Level Threads/KLT). Lebih aman.

  Contoh: Windows
- Gabungan, ada yang user/kernel space. Contoh: Solaris

ULT-KLT


Threads library
User space
Kernel space
P
(a) Pure user-level
User space
Kernel space
P
(b) Pure kernel-level
Threads library
User space
Kernel space
P
P
(c) Combined
User-level thread
Kernel-level thread
P
Process

Inter Process Communication

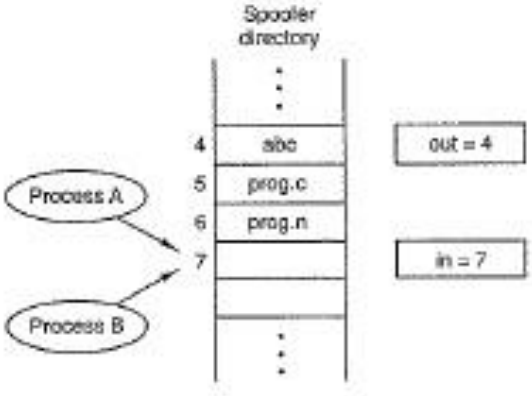
- Komunikasi yang terjadi antar proses
- Tiga masalah utama:
    1. Bagaimana caranya menyampaikan informasi
    2. Mencegah proses saling menghalangi
    3. Urutan aksi bila terjadi ketergantungan antar proses
- Juga berlaku untuk thread

Masalah IPC: Race Conditions

- Terjadi bila beberapa proses bersama-sama membaca/ menulis data, dan hasil akhirnya bergantung pada kapan suatu proses berjalan
- Contoh: printing file
  - **Spooler directory** untuk menuliskan nama file yang akan diprint
  - Terdiri atas sejumlah slot
  - **Printer daemon** memeriksa nama file yang akan diprint pada spooler directory dan menghapusnya
  - Dua variabel: out untuk mencatat slot berikutnya yang akan diprint dan in untuk mencatat slot berikutnya yang kosong

Printer Spooler

- Race condition bisa terjadi:
  - Proses A membaca in = 7 dan menyimpannya d variabelnya lalu waktunya habis
  - Proses B membaca in = 7 dan menyimpannya di variabelnya lalu menulis nama file di slot 7 (in menjadi 8) dan waktunya habis



  - Proses A kembali running dan menulis nama file di slot 7 (berdasarkan nilai di variabel lokalnya)
  - Akibatnya Proses B tidak pernah mendapatkan hasil printnya

IPC Primitives

- Busy waiting
- Sleep and wakeup
- Semaphore
- Mutex
- Monitor
- Message Passing
- Barrier

Critical Regions

- Adalah bagian program yang mengakses shared memory
- Perlu dibuat mutual exclusion, yaitu mencegah proses lain mengakses shared memory sebelum suatu proses selesai mengaksesnya
- Empat kondisi untuk menghindari race conditions:
  1. 2 proses tidak boleh bersamaan berada pada critical regions
  2. Tidak boleh diasumsikan kecepatan/banyaknya CPU
  3. Suatu proses yang sedang di luar critical region tidak boleh memblok proses lain
  4. Suatu proses tidak boleh menunggu selamanya untuk masuk critical region

Beberapa Upaya yang Gagal

- Men-disable interrupt saat masuk critical region kurang baik karena tidak seharusnya wewenang user process
- Menggunakan lock variable sama seperti printer spooler
- Strict alternation

```
while (TRUE) {
     while (turn != 0) ;
       critical_region();
     turn = 1;
     noncritical_region();
 }
```

```
while (TRUE) {
     while (turn != 1) ;
       critical_region():
   turn = 0;
   noncritical_region();
          }
```

Bagian loop disebut busy waiting

Masih melanggar kondisi 3, jika suatu proses yang cepat masuk ke critical region 2X berturut-turut

Mutual exclusion : Solusi Peterson

- Oleh G.I. Peterson (1981), menggantikan usulan Dekker

```
#define FALSE 0
#define TRUE 1
#define N   2                       /*number of processes*/
int turn;                           /*whose turn is it?*/
int interested[N];
void enter_region(int process) {    /*process is 0 or 1*/
   int other;                       /*id of the other process*/
   other = 1-process;               /*the opposite of process*/
   interested[process] = TRUE;      /*show your interest*/
   turn = process;                  /*set flag*/
   while (turn == process && interested[other] == TRUE) ; }
void leave_region(int process) {  /*process who is leaving*/
   interested[process] = FALSE; }
```

Mutual exclusion : Instruksi TSL

- TSL = Test and Set Lock
- Adalah instruksi (indivisible) untuk membaca shared variable lock ke register dan menuliskan suatu nilai (nonzero) ke lock
- Bila lock = 0 setiap proses boleh mengakses critical region dan menjadikan 0 lagi setelah selesai

```
enter_region:
   TSL REGISTER,LOCK        |copy lock to register and set to 1
   CMP REGISTER,#0          |was lock 0?
   JNE enter_region         |if it was nonzero then loop
   RET
leave_region:
   MOVE LOCK,#0             |store 0 to lock
   RET                      |return to caller
```

Mutual exclusion : Instruksi XCHG

```
enter_region:
   MOVE REGISTER,#1        |put a 1 in the register
   XCHG REGISTER,LOCK      |swap the content of the register and lock
   CMP REGISTER,#0         |was lock 0?
   JNE enter_region        |if it was nonzero then loop
   RET
leave_region:
   MOVE LOCK,#0            |store 0 to lock
   RET                     |return to caller
```

Mutual exclusion : Sleep and Wakeup

- Untuk menghindari busy waiting
- Selain memboroskan waktu CPU, juga bisa bermasalah jika ada prioritas proses
  - Proses dengan prioritas rendah (L) tidak akan bisa berjalan jika proses berprioritas tinggi (H) sedang berjalan
  - Saat L sedang dalam critical region, H melakukan busy waiting karena ingin masuk critical region
  - L tidak bisa keluar dari critical region (blocked) karena kalah prioritas
  - H tidak akan pernah masuk critical region
  - Disebut *priority inversion problem*

Masalah Producer-Consumer (1)

- Producer memasukkan informasi ke buffer (fixed size)
- Consumer mengambil informasi dari buffer
- Masalah: buffer penuh, diatasi dengan sleep dan wakeup system call
- Masih bisa tidur keduanya

Masalah Producer-Consumer (2)

Dapat terjadi tidur bersama (produser dan consumer.

```
#define N 100                                   f slots in the buffer*/
int count = 0;                                  /*number of items in the buffer*/
void producer(void) {
  int item;
  while (TRUE) {                                /*loop forever*/
     item = produce_item();                     /*generate next item*/
     if (count==N) sleep();                     /*if buffer is full go to sleep*/
     insert_item(item);                         /*put item into buffer*/
     count++;                                   /*increment count of items in buffer*/
     if (count==1) wakeup(consumer); } }        /*was buffer empty?*/
void consumer(void) {
  int item;
  while (TRUE) {                                /*loop forever*/
     if (count==0) sleep();                     /*if buffer is empty go to sleep*/
     item = remove_item();                      /*take item out of buffer*/
     count--;                                   /*decrement count of items in buffer*/
     if (count==N-1) wakeup(producer);          /*was buffer full*/
     consume_item(item); } }                    /*process item*/
```

Semaphore (1)

- Variabel untuk mencatat #(pending wakeup)
- 2 operasi down dan up yang indivisible: cek nilai, set nilai dan sleep jika nilai = 0
- Bisa diassign untuk tiap I/O device

Untuk mengatasi tidur bersama dengan semaphore

Semaphore (2)

```
#define N 100                    /*number of slots in the buffer*/
typedef int semaphore;                    /*semaphores are a special kind of integer*/
semaphore mutex = 1;                      /*control access to critical region*/
semaphore empty = N;                      /*counts empty buffer slots*/
semaphore full = 0;                       /*counts full buffer slots */
void producer(void) {
  int item;
  while (TRUE) {                          /*loop forever*/
     item = produce_item();               /*generate next item*/
     down(&empty);                        /*decrement empty count*/
     down(&mutex);                        /*enter critical region*/
     insert_item(item);                   /*put new item in buffer*/
     up(&mutex);                          /*leave critical region*/
     up(&full);    } }                    /*increment count of full slots*/
void consumer(void) {
  int item;
  while (TRUE) {                          /*loop forever*/
     down(&full);                         /*decrement full count */
     down(&mutex);                        /*enter critical region*/
     item = remove_item();                /*take item out of buffer*/
     up(&mutex);                          /*leave critical region */
     up(&empty);                          /*increment count of empty slots*/
     consume_item(item); } }              /*process item*/
```

Mutex (Mutual Exlusion)

- Adalah binary semaphore
- Cukup diimplementasikan dengan 1 bit

```
mutex_lock:
        TSL REGISTER,MUTEX      | copy mutex to register and set mutex to 1
        CMP REGISTER,#0         | was mutex zero?
        JZE ok                  | if it was zero, mutex was unlocked, so return
        CALL thread_yield       | mutex is busy; schedule another thread
        JMP mutex_lock          | try again
ok:     RET                     | return to caller; critical region entered

mutex_unlock:
        MOVE MUTEX,#0           | store a 0 in mutex
        RET                     | return to caller
```

Monitor

- Pemrograman dengan semaphore harus hati-hati
- Kesalahan urutan operasi semaphore bisa mengakibatkan deadlock
- Penanganan semaphore yang rumit diserahkan ke suatu konsep dalam bahasa pemrograman yang disebut monitor
- Monitor ditangani oleh compiler, bukan programmer
- Programmer hanya perlu memanggil monitor untuk masuk critical section

Producer dan Consumer dengan Monitor

```
monitor ProducerConsumer
    condition full, empty;
    integer count;

    procedure insert(item: integer);
    begin
        if count = N then wait(full);
        insert_item(item);
        count := count + 1;
        if count = 1 then signal(empty)
    end;

    function remove: integer;
    begin
        if count = 0 then wait(empty);
        remove = remove_item;
        count := count - 1;
        if count = N - 1 then signal(full)
    end;

    count := 0;
end monitor;
```

```
procedure producer;
begin
    while true do
    begin
        item = produce_item;
        ProducerConsumer.insert(item)
    end
end;

procedure consumer;
begin
    while true do
    begin
        item = ProducerConsumer.remove;
        consume_item(item)
    end
end;
```

Beberapa Masalah

- Monitor tidak dikenal di semua bahasa pemrograman
- Semaphore hanya berjalan untuk shared memory computer
- Bagaimana dengan distributed systems?
- Solusi: **message passing**

Message Passing

- 2 prosedur: send(dest,msg) dan receive(source,msg)
- Memerlukan protokol untuk masalah komunikasi data, mis: pesan yang hilang
- Beberapa isu: acknowledgement, sequence number, authentication

Producer and Consumer dengan Message Passing

```
#define N 100                                   /* number of slots in the buffer */

void producer(void)
{
     int item;
     message m;                                 /* message buffer */

     while (TRUE) {
          item = produce_item( );               /* generate something to put in buffer */
          receive(consumer, &m);                /* wait for an empty to arrive */
          build_message(&m, item);              /* construct a message to send */
          send(consumer, &m);                   /* send item to consumer */
     }
}
```

```
void consumer(void)
{
     int item, i;
     message m;

     for (i = 0; i < N; i++) send(producer, &m);  /* send N empties */
     while (TRUE) {
          receive(producer, &m);                  /* get message containing item */
          item = extract_item(&m);                /* extract item from message */
          send(producer, &m);                     /* send back empty reply */
          consume_item(item);                     /* do something with the item */
     }
}
```

Barrier (penghalang), menjaga

- Untuk menangani proses > 2
- Dibagi dalam beberapa fase
- Semua proses memasuki fase bersamaan
- Proses yang siap lebih dulu akan menunggu sampai semua proses lain siap

Sekian & terima kasih

Ada pertanyaan ?