ôn thi cuối học kỳ

Hệ điều hành (Trường Đại học Công nghệ thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh)

ĐỒNG BỘ TIẾN TRÌNH

1. Lý do:
    - Khảo sát các process/ thread thực thi đồng thời và chia sẻ dữ liệu(qua share mem, file)
    - Gây không nhất quán dữ liệu.
    - Cần cơ chế quản lý.

Bounded buffer: Consumer – producer

P ko đc ghi vào buffer đầy. C ko được đọc từ buffer trống. C,P ko đc đồng thời vào buffer

Đơn nguyên: (atomic), các thao tác như 1 lệnh đơn, không bị ngắt quãng giữa chừng.

Race condition: nhiều process cùng truy xuất và thao tác đồng thời lên dữ liệu chia sẻ.

Mutual exclusion (mutex): loại trừ tương hỗ: Khi một process vào cùng tranh chấp thì không có process khác thực thi các lệnh trong vùng tranh chấp.

Progress: Một tiến trình ngoài vùng tranh chấp không được ngăn process khác vào vùng tranh chấp.

Bounded waiting: chờ có giới hạn(tránh starvation: đói tài nguyên).

Nhóm giải pháp busy waiting: (giải pháp phần mềm: 3 cái đầu, phần cứng: 2 cái sau)

- Cờ hiệu
- Kiểm tra luân phiên
- Peterson
- Cấm ngắt
- Chỉ thị TSL

Nhóm giải pháp sleep and wake up:

- Semaphore
- Monitor
- Message

Busy waiting: tiêu thụ CPU khi đợi vào CS, không cần operating system.

Sleep and wake up: Từ bỏ CPU khi đợi, cần operating system.

BUSY WAITING: Giải pháp phần mềm

KIỂM TRA LUÂN PHIÊN:

Turn = i: Pi được vào CS (kiểm tra turn = I thì vào, xong thì gán lại = j)

Thoả Mutual exclusion nhưng không thoả progress và bounded waiting(khi P0 có remainder section lớn thì sẽ ngăn P1 có remainder section nhỏ vào CS)

CỜ HIỆU:

Plag[i] = true và Plag[j] = false thì process I được vào CS

Thoả mutual exclusion nhưng không thoả progress (khi p1 = true và p2 = true thì 2 process sẽ nhường nhau và không process nào chạy được)

PETERSON:
dùng cả cờ hiệu và kiểm tra luận phiên: Pi = true và turn = 1 mới được chạy

Thoả cả mutual exclusion, progress và bounded waiting

BAKERY: perterson cho n process

- Mỗi process nhận một con số.
- Số nhỏ nhất thì được vào CS.
- 2 process cùng số thì kiểm tra I < j thì I được vào CS.
- Cơ chế cấp số thường tạo theo cơ chế tăng dần.

Nhược điểm phần mềm: các process khi muốn vào CS thì phải liên tục kiểm tra điều kiện. Nếu một process có thời gian vào CS dài thì cần cơ chế block các process khác

BUSY WAITING: Giải pháp phần cứng.

CẤM NGẮT: DISABLE INTERUPTS

Chỉ tác dụng trên uniprocessor:mutex được đảm bảo

Tốn tài nguyên

Gây dừng đông hô

Không dùng được trên multiprocessor.

TEST AND SET: Đọc và ghi một biến đơn nguyên (atomic).

Dùng 1 biến lock = false, khi nào lock = false thì mới có tiến trình được vào và sẽ gán lại bằng true.

- Mutual exclusion được đảm bảo
- Tuỳ ý chọn process được thực hiện sau (có thể xảy ra starvation)
- Pentium dùng lệnh swap cũng có ưu điểm như test and set

SLEEP AND WAKE UP:
SEMAPHORE:

- Được cấp bởi Operating system, ko đòi hỏi busy waiting
- Là 1 biến số nguyên

- Ngoài thao tác khởi tạo biến thì chỉ có thể truy xuất qua tác vụ đơn nguyên và loại trừ (mutual exclusive)
  + wait: Giảm semaphore. Nếu sem âm thì Process gọi lệnh này bị block
  + signal: Tăng semaphore. Nếu sem không dương, thì một Process sẽ được phục hồi
- Tránh busy waiting: khi đợi thì process được đặt vào block queue,(sleep)
- Hàng đợi này là danh sách liên kết các PCB .
- Dùng cơ chế FIFO
- Block: running -> waiting, Wake up: waiting -> ready
- Đoạn mã định nghĩa wait(s) và signal(s) cũng là vùng tranh chấp(nhỏ: khoảng 10 lệnh).
- Giải pháp: dùng disable interrupt; dekker, Peterson, testandset, swap; CS nhỏ nên chi phí busy waiting thấp

Deadlock: chờ đợi vô hạn định một sự kiến không bao giờ xảy ra.

Starvation(indefinite blocking): Một Process có thể không bao giờ được lấy ra khỏi hàng đợi mà nó bị treo vào trong đó.

Các loại sem:

- Counting sem: một số nguyên có giá trị không hạn chế
- Binary sem: có giá trị là 1 hoặc 0 (dễ thực hiện).
- Có thể thực hiện couting sem = binary sem

BÀI TOÁN KINH ĐIỂN:

- Bounded buffer problem.(dùng 2 sem (full,empty) và 1 mutex)
- Dining – Philosopher problem.
- Readers and writers problem.

DINING – PHILOSOPHER: 5 triết gia ngôi ăn và có 5 chiếc đũa (tránh deadlock và starvation)

Giải pháp: dùng 5 sem, triết gia I dùng đũa thứ I và I + 1

Deadlock: cả 5 ng dùng đũa bên trái

Giải pháp: chỉ tối đa 4 triết gia được dùng đũa; chỉ khi nào đủ 2 đũa thì triết gia mới được câm đũa; triết gia lẻ câm đũa trái trước, triết gia chẵn câm đũa phải trước.

READERS AND WRITER: writer không được cập nhật khi reader đang truy xuất, writer không được cập nhật cùng nhau.

- Bộ đọc trước bộ ghi (first reader-writer)
- Dữ liệu chia sẻ

```
semaphore  mutex = 1;
semaphore  wrt   = 1;
int        readcount  = 0;
```

- Writer process

```
wait(wrt);
...
writing is performed
...
signal(wrt);
```

Reader process

```
wait(mutex);
readcount++;
if (readcount == 1)
        wait(wrt);
signal(mutex);
...
reading is performed
...
wait(mutex);
readcount--;
if (readcount == 0)
        signal(wrt);
signal(mutex);
```

Trong đoạn lệnh trên sem mutex dùng để thực thi đoạn lệnh một cách trọn vẹn, tránh bị ngắt quãng giữa chừng.

Wrt: được dùng bởi reader đầu hoặc cuối để tránh có writer vào ghi.

VẤN ĐỀ CỦA SEMAPHORE:

- Cung cấp công cụ để bảo đảm mutual exclusion và đồng bộ process.
- Khó năm bắt do wait và signal nằm rải rác. Gây deadlock hay starvation.
- 1 process bị die có thể kéo theo các process khác.

Giới thiệu thêm

CRITICAL REGION:

- Ngôn ngữ bậc cao, tiện cho người dùng
- Là 1 biến chia sẻ v kiểu T được khai báo: v: shared T;
- Được truy xuất: region v when B do S; (B là biểu thức boolean)
- Khi S được thực thi thì không có quá trình nào được truy xuất v

MONITOR:

- Là ngôn ngữ bậc cao, chức năng như sem nhưng dễ điêu khiển hơn
- Trong nhiêu ngôn ngữ lập trình : concurrent pascal, modula-3, java
- Có thể thực hiện = sem

Là một module phân mêm bao gôm:

- Một hay nhiêu thủ tục(procedure)
- Một đoạn code khởi tạo (initialization code)

- Các biến dữ liệu cục bộ (local data variable)

Đặc điểm:

- Local variable chỉ có thể truy xuất bởi procedure.
- Process vào monitor bằng cách gọi procedure.
- Chỉ 1 process có thể vào monitor vào một thời điểm.

Biến điều kiện (giống như lock): kiểm soát trong 1 lúc chỉ có 1 process được vào monitor

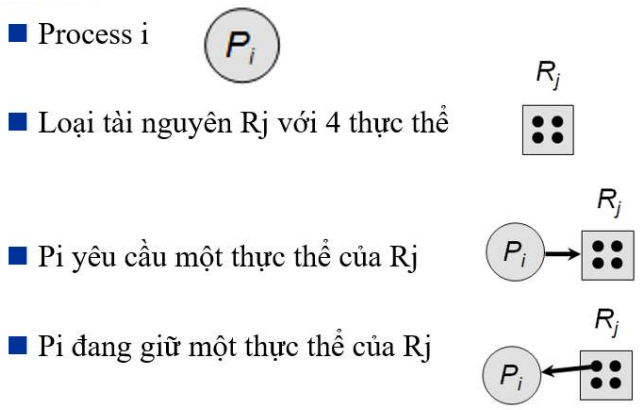
- Condition a,b
- Các biến điều kiện đều cục bộ và chỉ được truy cập trong monitor
- A.wait (lock), a.signal(unlock)
- Tiến trình sau khi gọi signal (unlock) sẽ được đưa vào urgent queue.

DEADLOCK:

Điều kiện deadlock:

- Loại trừ tương hỗ: ít nhất 1 tài nguyên được giữ theo non-shareable mode
- Giữ và chờ cấp thêm tài nguyên: Một tiến trình đang giữ tài nguyên và đang chờ tài nguyên do tiến trình khác giữ.
- Không trưng dụng: Tài nguyên không thể bị lấy lại mà chỉ có thể được trả khi process đó muốn
- Chu trình đợi: Tồn tại 1 chu trình đợi.

Đô thị cấp phát tài nguyên (RAG): tập đỉnh V, tập cạnh E.



Không chứa chu trình -> không có deadlock.

Chứa 1 hay nhiêu chu trình

- Mỗi tài nguyên chỉ có 1 thực thể: deadlock
- Mỗi tài nguyên có nhiêu thực thể: có thể xảy ra deadlock

Không cho hệ thống bị deadlock:

- Ngăn deadlock: Không cho phép 1 trong 4 điều kiện xảy ra deadlock
- Tránh deadlock: cung cấp thông tin để phân bổ tài nguyên hợp lý.

Cho hệ thống bị deadlock, phát hiện và phục hôi hệ thống.

Bỏ qua deadlock, xem như deadlock không tôn tại: hiện nay nhiêu hệ thống sử dụng; giảm dần hiệu suất. Cuối cùng ngưng hoạt động và khởi động lại.

NGĂN DEADLOCK:

- Mutual exclusion: không khả thi
- Hold and wait:
  + c1: Yêu cầu toàn bộ tài nguyên 1 lần. Nếu đủ thì cấp, không thì block
  + c2: Khi yêu cầu tài nguyên phải trả toàn bộ tài nguyên

- No preemption:
  + c1: Hệ thống lấy lại mọi tài nguyên đang giữ
  + c2: Hệ thống xem tài nguyên đang yêu câu : nếu tài nguyên A yêu câu nằm ở
    - Một tiến trình đang chờ tài nguyên -> lấy lại tài nguyên để cấp cho A
    - Một tiến trình không chờ tài nguyên -> Bắt A chờ và lấy lại các tài nguyên đang cần mà A đang cầm.
- Circular wait: Gán thứ tự cho tất cả tài nguyên trong hệ thống.

TRÁNH DEADLOCK:

- Ngăn deadlock sử dụng tài nguyên không hiệu quả.
- Đảm bảo hiệu suất sử dụng tài nguyên tối đa có thể.
- Yêu câu khai báo số lượng tài nguyên tối đa để thực hiện công việc
- Kiểm tra trạng thái cấp phát tài nguyên để đảm bảo hệ thống không rơi vào deadlock
- Trạng thái cấp phát dựa trên: TN còn, TN đã cấp, TN tối đa.

Gôm 2 giải thuật: Đô thị cấp phát tài nguyên(mỗi tài nguyên có 1 thực thể) và banker(có nhiêu thực thể).

BANKER:

Điêu kiện:

- Phải khai báo số lượng thực thể tối đa của từng tài nguyên
- Có thể phải đợi.
- Đã có đủ tài nguyên thì phải hoàn trả trong thời gian hữu hạn nào đó.

Phát hiện deadlock:

- Chấp nhận deadlock
- Phát hiện
- Phục hôi

PHÁT HIỆN:

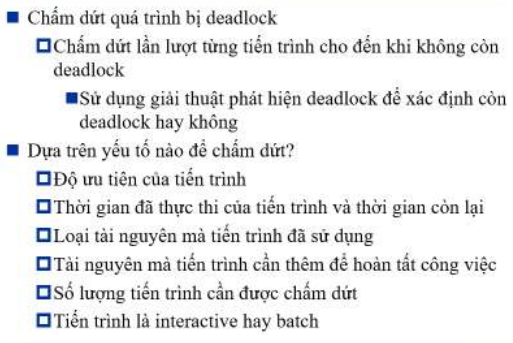
- Mỗi tài nguyên có 1 thực thể: dùng wait-for-graph Pi -> Pj
- Wait-for-graph được gọi định kỳ: Có chu trình -> có deadlock
- Có thời gian chạy là n^2 với n là số đỉnh

- Mỗi tài nguyên có nhiêu thực thể.
- Thời gian chạy $m*n^2$.

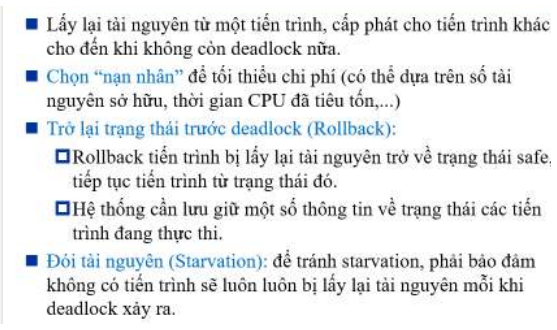
PHỤC HÔI: báo người vận hành; bẻ gãy chu trình.

- Chấm dứt tiến trình
- Lấy lại tài nguyên
- Kiểm lại xem còn deadlock hay không

Chấm dứt:



Lấy lại:



QUẢN LÝ BỌ NHỚ

Input queue: tiến trình trên đĩa chờ được mang vào ram để thực thi

User program trải qua nhiều bước trước khi xử lý.

Quản lý bộ nhớ: là công việc của hệ điều hành + phần cứng -> phân phối hiệu quả.

Mục tiêu: nạp càng nhiều process vào ram càng tốt

Kernel chiếm 1 phần ram, phần còn lại chia cho process

Các yêu câu với quản lý bộ nhớ:

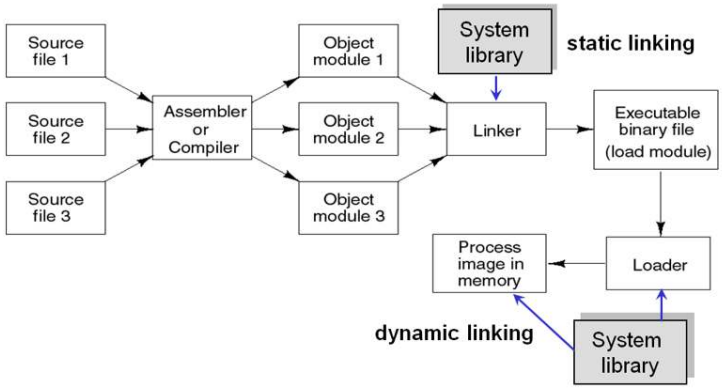
- Cấp phát bộ nhớ cho các process
- Tái định vị (relocate): khi swapping

- Bảo vệ: kiểm tra truy xuất bộ nhớ có hợp lệ hay không.
- Chia sẻ: cho phép các process chia sẻ vùng nhớ chung.
- Kết gán địa chỉ luận lý vào địa chỉ thực.

Địa chỉ luận lý (logical address): (còn gọi là virtual address) là một vị trí nhớ được diễn tả trong chương trình.

- Compiler: tạo ra mã lệnh -> địa chỉ luận lý.
- Địa chỉ tương đối (relative add): (relocatable add): là 1 kiểu logical address trong đó địa chỉ được biểu diễn tương đối so với 1 địa chỉ xác định trong chương trình.
- Địa chỉ tuyệt đối (absolute address) : địa chỉ tương đương với địa chỉ thực.

- Bộ linker: kết hợp các object module thành một file nhị phân khả thực thi gọi là load module.
- Bộ loader: nạp load module vào bộ nhớ chính



Chuyển đổi địa chỉ: ánh xạ một địa chỉ từ không gian địa chỉ này sang không gian địa chỉ khác.

Source code: symbolic (biến, hằng, pointer)

Thời điểm biên dịch: relocatable address.

Linking/loading: có thể là địa chỉ thực.

THỜI ĐIỂM CHUYỂN ĐỔI THÀNH ĐỊA CHỈ THỰC: 3 thời điểm:

- Compile time: Nếu biết trước địa chỉ bộ nhớ của chương trình
  VD: chương trình .com của MS-DOS. Khuyết điểm: phải biên dịch lại nếu thay đổi địa chỉ nạp.
- Load time: chuyển relocatable address thành địa chỉ thực dựa trên địa chỉ nền.
  Được tính toán lúc nạp chương trình. Phải reload nếu địa chỉ nền thay đổi.
- Execution time: Khi process có thể được di chuyển

Cần sự trợ giúp của phần cứng cho việc ánh xạ địa chỉ
Sử dụng trong đa số các OS đa dụng trong đó có các cơ chế swapping, paging, segmentation.

DYNAMIC LINKING:

- Thực thi khi đã tạo xong load module.
- Lúc này program cần liên kết đến module ngoài (external module)(cung cấp các tiện ích của OS, ko cần sửa hay biên dịch lại).
- Load module chứa các stub tham chiếu đến routine của external module.
- Stub cần sự hỗ trợ của OS (kiểm tra routine đã được nạp vào ram chưa, kiểm tra mã nào riêng, mã nào chung).
- Chia sẻ mã (code sharing): external module chỉ cần nạp vào ram 1 lần. Các process sẽ dùng chung mã này

DYNAMIC LOADING:

- Chỉ nạp khi cần thiết.
- Hiệu quả trong khối lượng mã lớn mà ít được dùng(vd: xử lý lỗi)
- Hỗ trợ từ OS: người dùng thiết kế và hiện thực, OS cung cấp thủ tục, thư viện

Mô hình quản lý bộ nhớ:

- Phân chia cố định (fixed partitioning)
- Phân chia động (dynamic partitioning)
- Phân trang đơn giản (simple paging)
- Phân đoạn đơn giản (simple segmentation)

PHÂN MẢNH NGOẠI (external fragmentation):

- Kích thước không gian nhớ đủ thoả một yêu câu cấp phát
- Không gian này không liên tục
- Dùng cơ chế kết khối (compaction) để gom lại thành vùng nhớ liên tục

PHÂN MẢNH NỘI (internal fragmentation):

- Kích thước không gian nhớ hơi lớn hơn yêu câu
- Xảy ra khi ram được chia thành khối cố định (fixed-size-block)
- Các process được cấp phát theo đơn vị khối (paging)

Fixed partitioning:

- Ram chia thành các phân rời nhau (gọi là partition) có kích thước bằng hay khác nhau .
- Process nào có kích thước <= partition thì được nạp vào partition đó.
- Nếu process > partition thì dùng overlay.
- Ko hiệu quả do bị phân mảnh nội

Chiến lược placement:

- Partition có kích thước bằng nhau:
  + Nếu còn partition thì nhét vào
  + Nếu không còn thì swap process đang bị block ra bộ nhớ phụ rôi nhét vào.
- Partition có kích thước không bằng nhau:
- 1.
  + Gán process vào partition nhỏ nhất phù hợp với nó.
  + Có hàng đợi cho mỗi partition (những process phù hợp xếp hàng chờ partition này)
  + Giảm thiểu phân mảnh nội.
  + Vấn đề: partition trống không có process đợi, partition lại nhiều process đợi.
- 2.
  + Dùng 1 hàng đợi chung.
  + Nạp process vào partition nhỏ nhất còn trống

DYNAMIC PARTITIONING

- Số lượng partition không cố định và có thể có kích thước khác nhau
- Cấp phát chính xác dung lượng ram yêu câu.
- Gây ra hiện tượng phân mảnh ngoại.

Chiếc lược placement:

- Mục tiêu: giảm chi phí compaction
- Chiến lược: best-fit, first-fit, next-fit, worst-fit

SIMPLE PAGING:

- Bộ nhớ vật lý -> frame
- Logical memory: -> page
- Page table: ánh xạ page thành frame

Logical memory: page num + Page offset

PTBR: trỏ đến bảng phân trang

PTLR: biểu thị kích thước bảng trang

Cache tìm kiếm: TLBs (translation look-aside buffers)

EAT = (2 – α)x + ε trong đó: α là tỉ lệ tìm thấy trong TLB

X là thời gian một chu kì truy xuất trong bộ nhớ

ε là thời gian tìm thấy trong TLB

Tổ chức bảng trang: Bảng trang nghịch đảo (vd: IBM) sử dụng cho tất cả process

Bảo vệ bộ nhớ: Gắn với frame các bit bảo vệ. Gôm read-only, write-only, execute-only

Và 1 valid/invalid bit gắn với mỗi mục trong bảng phân trang.

Swapping (cơ chế hoán vị): swapping policy: round -robin; roll out, roll in.

BỘ NHỚ ẢO

Virtual memory: Là kĩ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý.

Ưu điểm:

- Số process trong bộ nhớ nhiêu hơn
- Một process có thể được thực thi ngay cả khi kích thước nó lớn hơn bộ nhớ thực
- Giảm nhẹ công việc của lập trình viên

Swap space: Không gian tráo đổi giữa bộ nhớ chính và bộ nhớ phụ.

Vd: swap partition trong linux

File pagefile.sys trong windows.

Kỹ thuật: 2 kỹ thuật

- Phân trang theo yêu câu (demand paging)
- Phân đoạn theo yêu câu (demand segmentation):

Phần cứng hỗ trợ 2 kỹ thuật này.

OS quản lý sự di chuyển giữa ram và bộ nhớ thứ cấp

PHÂN TRANG THEO YÊU CÂU (DEMAND PAGING):

- Các trang được nạp vào ram khi được yêu câu
- Khi tham chiêu đến trang ko có trong ram -> ngắt (page-fault-trap) -> khởi tạo page-fault service routine (PFSR) của OS
- PFSR
  + chuyển process vê trạng thái block
  + yêu câu đọc đĩa để nạp trang cần vào frame trống; trong khi đợi I/O, process khác được cấp CPU để thực thi
  + Sau khi I/O, đĩa gây ra ngắt đến OS, PFSR cập nhật page table và chuyển process vê ready.

Nghịch lý Belady: số page-fault tăng mặc dù đã cấp nhiêu frame hơn

Cấp phát frame cho process.

- Cấp phát tĩnh (fixed-allocation): số frame cấp ko đổi, dựa vào thời điểm loading và kích thước của nó
- Cấp phát động (variable-allocation): dựa vào tỉ lệ page-fault và thay đổi khi chạy

Fixed-allocation: gôm cấp phát bằng nhau, cấp phát theo tỉ lệ kích thước process, cấp phát theo độ ưu tiên.

Thrashing: hiện tượng các trang nhớ của một process bị hoán chuyển vào/ra liên tục.

Nguyên lý locality (locality principle): thrashing xuất hiện khi Σ size of locality > memory size

Giải pháp tập làm việc:

- Thiết kế dựa trên locality principle.
- Xác định process thực sự sử dụng bao nhiêu frame.
- WS(t): tập các trang nhớ được truy xuất trong khoảng thời gian t - denta
- WSS(t): số lượng tham chiếu trang nhớ

Đặt D = Σ WSSi = tổng các working-set size của mọi process trong hệ thống.

- Nhận xét: Nếu D > m (số frame của hệ thống) ⇒ sẽ xảy ra thrashing.

Giải pháp working set:

- Khi tạo 1 process: cấp cho nó một số frame thoả woking set size của nó
- Khi D > m -> tạm dừng 1 trong các process
- Các trang được chuyển ra đĩa cứng và thu hôi các frame của nó
- WS loại trừ tình trạng trì trệ mà vẫn đảm bảo mức độ đa chương.