**Operating Systems**

Εισαγωγή

Το Λειτουργικό Σύστημα (ΛΣ) είναι ένα πρόγραμμα - διεπαφή ανάμεσα στο υλικό ενός υπολογιστικού συστήματος και τους χρήστες του (εφαρμογές).

**Ανάγκη ύπαρξης** ΛΣ:

* διαχείριση διεργασιών (pipes, αρχεία) – νημάτων
* συγχρονισμό, απομόνωση και ασφάλεια διεργασιών
* διαχείριση μνήμης (φυσικής - εικονικής)
* διαχείριση αρχείων
* επιτρέπει την επικοινωνία με συσκευές εισόδου/εξόδου (ποντίκι, πληκτρολόγιο)
* επιτρέπει τη δικτύωση του υπολογιστή (IP address, WIFI)
* επικοινωνία με τον χρήστη (command line/ user interface)

Μηχανισμούς υλικού (hardware) που υποστηρίζουν:

1. το διαχωρισμό της εκτέλεσης του ΛΣ από τα προγράμματα χρήστη
   1. με προνόμια -> κάποιες διεργασίες πρέπει να εκτελούνται μόνο από ΛΣ
   2. γίνεται διαχωρισμός του επεξεργαστή σε δύο καταστάσεις kernel-user
   3. αυτό γίνεται με τη δέσμευση ενός bit σε έναν καταχωρητή -> bit mode
2. τη χρονοδρομολόγηση
   1. ο καταχώρησης που λειτουργεί ως counter και μετρά το κβάντο χρόνου, χρόνο που βρίσκεται μία διεργασία τη CPU
3. την εικονική μνήμη
   1. TLB (Translation Lookaside buffer) είναι μία cache μνήμης που βοηθάει στη μετάφραση διευθύνσεων
   2. εκεί αποθηκεύονται οι διευθύνσεις των πιο πρόσφατων προσπελάσεων στο page table (μαζί με τις γειτονικές τους διευθύνσεις – γιατί έχουν μεγάλη πιθανότητα να προσπελαστούν στη συνέχεια)
   3. page table register -> που αποθηκεύει τη πρώτη διεύθυνση ου page table για να μπορείς να βρεις πού είναι
4. το συγχρονισμό
   1. οι ατομικές εντολές

Το ΛΣ εμπλέκεται στις παρακάτω περιπτώσεις:

1. το πρόγραμμα του χρήστη καλεί μία συνάρτηση/υπορουτίνα
   1. σε περίπτωση που η συνάρτηση είναι ή περιέχει κλήση συστήματος (system call)
   2. αν η διεργασία δε τελειώσει εγκαίρως στο κβάντο χρόνου της
2. αν συμβεί page fault και ο κώδικας οδηγεί σε εξαίρεση null pointer dereference
3. δεν έχω άδεια πρόσβασης/exception (π.χ. διαίρεση με το 0, null pointer)
4. συμβαίνει ένα cache miss
   1. αν η σελίδα βρίσκεται στο page table τότε το ΛΣ δεν εμπλέκεται
   2. αν όχι τότε εμπλέκεται στη μεταφορά της από το δίσκο
5. συμβαίνει ένα TLB (Translation Lookaside buffer) miss
   1. τότε επιχειρείται ανάγνωση του page table
   2. το ΛΣ εμπλέκεται αν γίνει page fault
6. συμβαίνει ένα σφάλμα σελίδας (page fault)
   1. ελέγχεται το TLB -> μετά το page table -> τότε page fault
   2. το ΛΣ ξυπνάει αφού έγινε αναφορά σε άκυρη σελίδα (invalid)
   3. το ΛΣ συμβουλεύεται το χάρτη μνήμης και έλεγχος δικαιωμάτων
      1. μη επιτρεπόμενη πρόσβαση -> segmentation fault
      2. επιτρεπόμενη πρόσβαση αλλά invalid
         1. η σελίδα είναι στο δίσκο
         2. ο χρονοδρομολογητής βάζει την διεργασία σε WAITING και τη βγάζει από τη CPU συνεχίζοντας με την επόμενη που βρίσκεται στην ουρά αναμονής
         3. μεταφορά από το δίσκο
         4. ανάγνωση δεδομένων
         5. ανανέωση χάρτη μνήμης
         6. ανανέωση page table (invalid -> valid)
         7. χρονοδρομολογητής -> READY state
         8. διακόπτεται η CPU και εκτελείται
7. το πρόγραμμα του χρήστη κάνει διαίρεση με το 0
   1. το ΛΣ στέλνει segmentation fault και σκοτώνει τη διεργασία εκτός αν έχουμε signal handler
8. το πρόγραμμα του χρήστη κάνει null pointer dereference
   1. το ΛΣ στέλνει segmentation fault και σκοτώνει τη διεργασία εκτός αν έχουμε signal handler
9. το πρόγραμμα του χρήστη κάνει άλμα σε χώρο μνήμης που περιέχει δεδομένα
   1. θα γίνει page fault και το ΛΣ θα τερματίσει το πρόγραμμα αφού σε αυτό το χώρο δε θα υπάρχει δικαίωμα “x”
10. το πρόγραμμα του χρήστη προσπαθεί να μπει σε κρίσιμο τμήμα
    1. το ΛΣ εμπλέκεται στη περίπτωση που χρησιμοποιούνται semaphores /mutexes και το κρίσιμο τμήμα δεν είναι διαθέσιμο με τη wait/wakeup οπότε η διεργασία κάνει sleep και χρονοδρομολογείται άλλη διεργασία

\*\*\* **Μια διεργασία δεν έχει απευθείας πρόσβαση στο υλικό**

π.χ. **ld r1, [r2]** μεταφερει το περιεχομενο της θεσης μνημης στην οποία δειχνει ο καταχωητης r2 στον καταχωρητη r1 και οδηγει:

1. στην εμπλοκη του ΛΣ δυο φορες

Page fault -> μεταφορά από το δίσκο -> εκτέλεση

2. τερματισμός διεργασίας

χάρτης μνήμης -> έλεγχος δικαιωμάτων -> δεν έχουμε άδεια για read στο r2 -> οπότε το ΛΣ τερματίζει τη διεργασία

3. το πρόγραμμα συνέχισε

χάρτης μνήμης -> έλεγχος δικαιωμάτων -> δεν έχουμε άδεια για read στο r2 -> invalid -> οπότε μεταφορά από το δίσκο -> valid στο χάρτη μνήμης -> εκτέλεση εντολής

Συχνά λέμε ότι το εκτελέσιμο που προκύπτει από τη μεταγλώττιση ενός προγράμματος μπορεί να εκτελεστεί μόνο σε συγκεκριμένο ΛΣ (π.χ. linux). Τι σημαίνει αυτο?

Εκτελέσιμο που μεταγλωττίζεται για το Linux δεν θα εκτελεστεί σωστά σε άλλα λειτουργικά συστήματα, όπως τα Windows ή το macOS, λόγω των διαφορετικών τεχνολογιών και διεπαφών που χρησιμοποιούνται από αυτά τα λειτουργικά συστήματα ->προβλήματα στις κλήσεις του συστήματος (system calls)

Διεργασίες & Νήματα

**Διεργασίες:**

* είναι ένα ζωντανό πρόγραμμα που εκτελείται ΤΩΡΑ
* για να γίνει ζωντανό πρέπει να έχει
  + μνήμη
  + αρχιτεκτονική κατάσταση (στιγμιότυπο μίας διεργασίας στη CPU, οι τιμές των καταχωρητών της)
* Κάθε διεργασία έχει τη δικιά της μνήμη -> τον δικό της χώρο διευθύνσεων μνήμης, πίνακα προγραμμάτων και πόρους, όπως αρχεία και συσκευές εισόδου/εξόδου.
* Μπορεί να περιέχει πολλά νήματα
* Είναι απομονωμένες μεταξύ τους
  + δεν μπορούν να αλληλεπιδρούν άμεσα μεταξύ
* Ασφάλεια: Δε μοιράζονται μνήμη οπότε η παραβίαση των δεδομένων είναι δυσκολότερη

**Νήματα:**

* Είναι ένα εκτελέσιμο μέρος μιας διεργασίας
* Έχει κοινή μνήμη με τα υπόλοιπα νήματα της ίδιας διεργασίας
  + μπορούν να επικοινωνούν μεταξύ τους ευκολότερα
  + μοιράζεται τον ίδιο χώρο διευθύνσεων μνήμης, πίνακα προγραμμάτων και πόρους
* Απόδοση: Τα νήματα είναι πιο ελαφριά και γρήγορα στη δημιουργία και την εναλλαγή από τις διεργασίες.

Αναφερεται τουλάχιστον 5 λογους για τους οποίους μία διεργασία ΔΕΝ ολοκλήρωσε το κβάντο χρόνου της :

1. Ολοκληρώθηκε νωρίτερα
2. Στάλθηκε signal που την έκανε kill από το πρόγραμμα
3. Μπορεί να κάνει wait μέχρι να διαβάσει κάτι
4. Segmentation fault -> διαίρεση με το 0/ null pointer dereference
5. Μπορεί να ολοκληρώθηκε η μεταφορά διεύθυνσης μνήμης από το δίσκο και χρονοδρομολογητής να την έκανε wait
6. Permission error -> να προσπάθησε να διαβάσει κάτι που δεν έχει άδεια

Αναφέρετε τι πληροφορία κρατάει ο Πίνακας ελέγχου Διεργασίας(PCB) που να υποστηρίζει:

* Γενικά περιέχει:
  + Ταυτότητα διεργασίας (PID)
  + State
  + PC – registers
  + CPU scheduling info (προτεραιότητες)
    - FCFS, SJF, RR, Priority Scheduling
    - Kριτήρια αξιολόγησης: throughput, response time, turnaround time(ολοκλήρωσης), waiting time, CPU utilization(χρησιμοποίηση)
  + Memory management info (δεσμευμένες περιοχές μνήμης, είδος, όρια πρόσβασης)
  + Accounting info (time, CPU usage)
  + Πληροφορίες για τον έλεγχο πρόσβασης(ταυτότητα χρήστη, δικαιώματα πρόσβασης, δυνατότητες)
  + Current directory
  + I/O status info (πίνακας ανοιχτών αρχείων)
* τη μετάφραση των λογικών διευθύνσεων σε φυσικές
  + χάρτης μνήμης
* τη πρόσβαση στο Σύστημα Αρχείων
  + current directory
  + Local πίνακας Ανοικτών αρχείων: Μια δομή δεδομένων που καταγράφει τα ανοιχτά αρχεία της διεργασίας. Κάθε ανοιχτό αρχείο έχει αντιστοιχιστεί ένας κατάλογος αρχείου που περιέχει πληροφορίες όπως η θέση του δείκτη αρχείου και τα δικαιώματα πρόσβασης.
* την εναλλαγή περιβάλλοντος (context-switch)

*[είναι η διαδικασία στην οποία το ΛΣ αλλάζει τη διεργασία που εκτελείται στη CPU, αποθηκεύει τη κατάσταση της τρέχουσας διεργασίας στο PCB της ή την επαναφέρει]*

* + στιγμιότυπο της αρχιτεκτονική κατάσταση της τρέχουσας διεργασίας
  + δηλαδή αντίγραφα των τιμών όλων των καταχωρητών της για να μπορεί να την επαναφέρει

*Η εναλλαγή περιβάλλοντος ανάμεσα σε νήματα της ίδιας διεργασίας είναι φθηνότερη, γιατί όλα τα νήματα τρέχουν στο ίδιο περιβάλλον μνήμης 🡪 δε χρειάζεται αλλαγή της κατάστασης της MMU*

Μία διεργασία θέλει να ενημερώσει μία άλλη διεργασία για την τιμή μίας μεταβλητής. Αναφέρετε δύο τρόπους με τους οποίους μπορεί να γίνει αυτό.

1. Κοινή μνήμη (όπως fork())
2. Pipes

Τι χαρακτηριστικά πρέπει να έχει ένας κώδικας (ή αλγόριθμος) ώστε η ταυτόχρονη εκτέλεσή του από πολλές διεργασίες (ή νήματα) να μπορεί να οδηγήσει σε ασυνέπειες; Αναφέρετε ένα παράδειγμα.

Αν υπάρχουν πολλαπλές διεργασίες που εκτελούν εργασίες σε κοινά δεδομένα και ο χρήστης δεν έχει προνοήσει για κατάλληλο συγχρονισμό αυτών, δηλαδή οι διεργασίες να εισέρχονται μέσω αμοιβαίου αποκλεισμού στο κρίσιμο τμήμα και όχι ταυτόχρονα. τότε μπορεί να υπάρξουν ασυνέπειες. Χαρακτηριστικό παράδειγμα η αυξομείωση μιας ακέραιας μεταβλητής που ακολουθεί το μοντέλου παραγωγού-καταναλωτή. Αν επιτρέπεται στις διεργασίες να εισάγονται ταυτόχρονα στο κρίσιμο τμήμα ( αύξηση/μείωση της μεταβλητής ) τότε η μία δεν γνωρίζει την μεταβολή που θα επιφέρει η άλλη με συνέπεια στο τελικό επιθυμητό αποτέλεσμα.

Το κλείδωμα (lock) δίνει μία λύση στο πρόβλημα του Κρίσιμου Τμήματος. Εξηγήστε πως ανάλογα με την υλοποίηση ενός κλειδώματος μπορεί να είναι απαραίτητη ή μη απαραίτητη η συνδρομή του Λειτουργικού Συστήματος.

Το ΛΣ χρειάζεται να επέμβει για να κοιμίζει/ξυπνάει διεργασίες αν πρόκειται για semaphores/mutexes, ενώ η συμβολή του δεν απαιτείται στην περίπτωση των spinlocks όπου υλοποιούνταιι με την ενεργός αναμονή(busy wait).

Ένα πρόγραμμα (διεργασία) υπό εκτέλεση θέλει να διαβάσει δεδομένα από το σκληρό δίσκο μέσω Απευθείας Προσπέλασης Μνήμης (Direct Memory Access - DMA). Περιγράψτε τα συμβάντα (κλήσεις συστήματος, διακοπές, κλπ), τις λειτουργίες του Λειτουργικού Συστήματος και τις καταστάσεις από τις οποίες περνά η διεργασία μέχρι να ξαναβρεθεί σε κατάσταση υπό εκτέλεση.

Μια διεργασία κάνει αίτηση για δεδομένα από τον δίσκο και ακολουθείται η παρακάτω διαδικασία:

1. Στέλνεται αίτημα από το ΛΣ και η διαδικασία αφαιρείται από τη CPU και τίθεται σε Waiting state
2. Η CPU μπορεί συνεχίσει τη λειτουργία της με διεργασίες από την ουρά έτοιμων διεργασιών
3. Τα δεδομένα από τον δίσκο μεταφέρονται σε καταχωρητές
4. Μόλις ολοκληρωθεί η μεταφορά στέλνεται interrupt στη CPU
5. Η διεργασία που ζήτησε τα δεδομένα τίθεται από το ΛΣ σε Ready state και τοποθετείται πάλι στην ουρά έτοιμων διεργασιών ώστε να επιλεχθεί από τον ΧΔ και να τεθεί υπό εκτέλεση από το σημείο που είχε διακοπεί.

Αναφέρετε ένα παράδειγμα κώδικα ή αλγορίθμου που λειτουργεί σωστά για μία διεργασία και λανθασμένα για περισσότερες από μία διεργασίες. Εξηγήστε.

void increase\_fn(int \*var) {

for(i=0;i<n;i++)

++(\*var);

}

void decrease(int \*var) {

for(i=0;i<n;i++)

--(\*var);

}

Όταν εκτελεστεί ο παραπάνω κώδικας από μια διεργασία , επειδή η πρόσβαση στα δύο ΚΤ (++(\*var) & --(\*var) ) θα γίνει σειριακά , το τελικό αποτέλεσμα θα είναι \*var=0. Αν όμως εκτελεστεί από περισσότερες διεργασίες θα υπάρχουν ασυνέπειες επειδή δεν έχουμε φροντίσει για τον συγχρονισμό αυτών. Ως αποτέλεσμα οι διεργασίες θα εισέρχονται ταυτόχρονα στα ΚΤ πραγματοποιώντας τυχαία αυξομειώσεις της \*var χωρίς να λαμβάνουν υπ’οψιν τις υπόλοιπες και επηρεάζοντας έτσι το τελικό αποτέλεσμα.

Μία διεργασία προκαλεί σφάλμα σελίδας και το ΛΣ θα πρέπει να τη χειριστεί. Αναφέρεται δύο διαφορετικές αιτίες που μπορεί να οδηγήσουν σε σφάλμα σελίδας και περιγράψτε σύντομα τι θα πρέπει να κάνει το ΛΣ σε κάθε περίπτωση.

* 1. Η διεργασία προσπαθεί να προσπελάσει διεύθυνση εικονικής μνήμης που δε της ανήκει. Τότε το ΛΣ πρέπει να στείλει κατάλληλο σήμα στη διεργασία για να τη τερματίσει.
  2. Η διεργασία προσπαθεί να προσπελάσει διεύθυνση εικονικής μνήμης που της ανήκει, αλλά έχει μεταφερθεί στο δίσκο. Τότε, το ΛΣ πρέπει να φέρει την αντίστοιχη σελίδα από το δίσκο και να φτιάξει τις κατάλληλες απεικονίσεις(mappings), ώστε η σελίδα να είναι διαθέσιμη στην εργασία. Όταν γίνει αυτό η διεργασία τίθεται σε κατάσταση ‘ready’

Χρονοδρομολόγηση

*Ο τρόπος με τον οποίο μια διεργασία θα φύγει απο τη CPU και θα έρθει μια άλλη στη θέση της - αλλαγή περιβάλλοντος (context switch)*

Ο χρονοδρομολογητής σε ένα σύστημα με πολλούς επεξεργαστές μπορεί να τηρεί μία ουρά έτοιμων διεργασιών ανά επεξεργαστή, ή μία καθολική ουρά. Αναφέρετε μειονεκτήματα και πλεονεκτήματα της κάθε προσέγγισης.

Κάθε επεξεργαστής διατηρεί ξεχωριστή ουρά έτοιμων διεργασιών:

* Απλή υλοποίηση
* Ελάχιστος χρόνος αναμονής
* αποφεύγεται η μετακίνηση διεργασιών μεταξύ τον επεξεργαστών
* Οδηγούμαστε, όμως, σε προσκόλληση διεργασιών σε επεξεργαστή (processor affinity)
* κάποιος επεξεργαστής μπορεί να μείνει ανενεργός για μεγάλο διάστημα, εφ‘ όσον δεν του ανατίθενται καινούριες διεργασίες.

Η καθολική ουρά:

* + μπορεί να το αντιμετωπίσει
  + χρειάζεται όμως προσοχή στον δρομολογητή ώστε να αποφεύγεται η ανάθεση της ίδιας διεργασίας σε πολλούς επεξεργαστές.
  + Επίσης δημιουργούνται κοινά δεδομένα στις caches των επεξεργαστών, αφού λειτουργούν την ίδια διεργασία (σε διακοπτό χρονοδρομολογητή).

Μία διεργασία έκανε σφάλμα σελίδας και στο τέλος της διαδικασίας ενεπλάκη ο χρονοδρομολογητής. Αναφέρετε τα ενδιάμεσα βήματα από το σφάλμα σελίδας μέχρι την εμπλοκή του χρονοδρομολογητή.

* 1. Το ΛΣ συμβουλεύεται το χάρτη μνήμης
  2. Η σελίδα βρίσκεται στο δίσκο
  3. Έλεγχος δικαιωμάτων πρόσβασης
  4. Διακοπή τη διεργασίας από το ΛΣ λόγω page fault.
  5. Αποθήκευση του PCB της και αλλαγή σε Waiting-state επειδή η μεταφορά της από το δίσκο είναι χρονοβόρα
  6. Ο χρονοδρομολογητής αναθέτει άλλη διεργασία στην CPU από την ουρά έτοιμων διεργασιών
  7. Αναζήτηση στη μνήμη για ελεύθερο πλαίσιο. Αν υπάρχει φόρτωση της καινούριας σελίδας. Αν δεν υπάρχει εύρεση πλαισίου για αντικατάσταση (FIFO/LRU)
  8. Ανανέωση πίνακα σελίδων της διεργασίας
  9. Αλλαγή σε Ready-state και αναμονή για την ανάθεση στην CPU από τον ΧΔ

**Non-preemptive – Preemptive**

Ένας non-preemptive χρονοδρομολογητής:

* εκτελείται όταν μια διεργασία μεταβεί σε κατάσταση αναμονής ή όταν τερματίσει.
* Απλός - Μικρότερη πολυπλοκότητα
* Οι εκτελούσες διεργασίες αποφασίζουν πότε θα προχωρήσουν την εκτέλεση σε άλλες διεργασίες
* λαμβάνει απόφαση χρονοδρομολόγησης μόνο στις μεταβάσεις:
  + RUNNING 🡪WAITING, RUNNING 🡪 TERMINATED
  + Που αντιστοιχούν σε διακοπές λογισμικού: η τρέχουσα διεργασία προκαλεί trap για κλήση συστήματος ή pagefault
* Ανεπάρκεια σεναρίων που απαιτούν δίκαιη και ισότιμη κατανομή του χρόνου CPU
  + Εάν μια διεργασία δεν παραχωρήσει εθελοντικά τον έλεγχο, άλλες διεργασίες μπορεί να μην έχουν ποτέ την ευκαιρία να εκτελεστούν.
  + ανεπίτρεπτα μεγάλοι χρόνοι αναμονής για ορισμένες διεργασίες
* κατάλληλος για υπερυπολογιστές

Ένας preemptive χρονοδρομολογητής:

* μπορεί να διακόπτει τις διεργασίες
* λαμβάνει απόφαση χρονοδρομολόγησης στις μεταβάσεις:
  + RUNNING 🡪WAITING, RUNNING 🡪 TERMINATED
  + RUNNING 🡪READY, WAITING 🡪 READY
    - Που αντιστοιχούν σε διακοπές υλικού από εξωτερικούς παράγοντες:
      * ο χρονιστής διακόπτει τη CPU γιατί το κβάντο χρόνου τελέιωσε
      * ο δίσκος διακόπτει τη CPU γιατι μία μεταφορά ολοκληρώθηκε
    - μπορούν να οδηγήσουν σε αλλαγή της τρέχουσας διεργασίας και με force
* προσφέρει τη δυνατότητα προτεραιότητας και ελέγχου διεργασιών
* αποτελεσματική διαχείριση του χρόνου CPU
* καλύτερη αποκρισιμότητα
* διασφάλιση ότι κάθε διεργασία θα έχει ευκαιρία για εκτέλεση
* πρόσθετη πολυπλοκότητα 🡪 διαχείριση διακοπτών/προτεραιοτήτων
* προσωπικός υπολογιστής

**Χρονοδρομολόγηση FCFS (First-Come, First-Served)**

* Απλός, εύκολος και δίκαιος στην υλοποίηση.
* Δίνει προτεραιότητα στις διεργασίες που έφτασαν πρώτες.
* Δεν λαμβάνει υπόψη τον χρόνο εκτέλεσης μιας διεργασίας, οπότε μπορεί να προκαλέσει μεγάλους χρόνους αναμονής για πολύ υπολογιστικά απαιτητικές διεργασίες -> για να το μειώσω **ShortestJobFirst**
* Δεν έχει ανάγκη σε υλικό

**Χρονοδρομολόγηση RR (Round Robin)**

* Δίκαιη κατανομή του χρόνου επεξεργασίας σε όλες τις διεργασίες με ορισμένο κβάντο χρόνου (μέσω καταχωρητή μετρητή)
* Κυκλική πρόσβαση στη CPU
* Προσωπικός υπολογιστής:
  + ψευδαίσθηση παράλληλης εκτέλεσης πολλαπλών διεργασιών
  + χαμηλός χρόνος αποκρισιμότητας
* Υπάρχει overhead λόγω της συχνής εναλλαγής διεργασιών
  + ο χρονοδρομολογητής πρέπει να εκτελείται συχνά context-switch
  + αποθήκευση κάθε φορά σε καταχωρητές της κατάστασης κάθε διεργασίας για να συνεχιστεί στη συνέχεια
* Μπορεί να προκαλέσει αυξημένο χρόνο αναμονής για μακροχρόνιες διεργασίες 🡪 εκχωρείται ίδιο χρονικό κβάντο σε όλες όποτε πιθανόν να περιμένουν για αρκετές εκτελέσεις πρωτού τους δοθεί πάλι η σειρά να εκτελεστούν

**Χρονοδρομολόγηση ShortestJobFirst:**

* επιλέγεται η διεργασία με το μικρότερο χρόνο burst CPU (βέλτιστος)
* μη υλοποιήσιμος αφού γενικά το μήκος επόμενου burst δεν ειναι γνωστό
  + - * + προσέγγιση με βάση προηγούμενες τιμές – ξεκινάω με έναν random χρόνο εκτέλεσης
      * φαινόμενο λιμοκτονίας --> χρησιμοποιούμε Γήρανση (aging) με την πάροδο του χρόνου αυξάνεται η προτεραιότητα των διεργασιών που δεν εκτελούνται

Ο χρονοδρομολογητής βρίσκει στην ουρά έτοιμων διεργασιών μόνο μία διεργασία την οποία και χρονοδρομολογεί για ένα κβάντο χρόνου. Στο τέλος του κβάντου χρόνου, ο χρονοδρομολογητής βρίσκει στην ουρά έτοιμων διεργασιών Ν διεργασίες. Αναφέρετε τρία πράγματα που μπορεί να συνέβησαν κατά τη διάρκεια του κβάντου χρόνου κάτα το οποίο εκτελέστηκε η αρχική διεργασία.

* + - * Η διεργασια μπορει να ξυπνησε αλλες διεργασιες
      * Μπορει ο χρηστης να ανοιξε αλλες εφαρμογές που εφεραν αλλες Ν διεργασιες
      * Μπορει να υπήρχαν σε sleep mode (wait) και το τέλος αυτής της διεργασίας να τις ξύπνησε

Με ποιο τρόπο (σχήμα χρονοδρομολόγησης) μπορεί το ΛΣ να αποκτήσει μεγαλύτερο έλεγχο στο χρόνο αναμονής των διεργασιών;

Με πολυεπίπεδες ουρές αναμονής

Συγχρονισμός

Πρόβλημα παραγωγός-καταναλωτής -> λύνεται με 3 mutexes (space, items, mutex)

Κρίσιμο τμήμα

* επιτρέπει μόνο μία διεργασία κάθε φορά
* Προστατευόμενο τμήμα/ εκτελείται ατομικά
  + 1. lock/unlock -> Λύση Peterson
       - δε μπορεί να χρησιμοποιηθεί γιατί υλοποιείται μόνο για 2 διεργασίες και ο CPU κάθε φορά μπορεί να αναδιατάσσει τις εντολές του προγράμματος και αυτές να μην εκτελούνται με τη σωστή σειρά
    2. ατομικές εντολές
       - λειτουργούν ατομικά
       - εκτελούνται από το υλικό
    3. spinlocks
       - ζητούν συνεχώς να μπουν στο κρίσιμο τμήμα μέχρι να το εκτελέσουν (ενεργός αναμονή)
       - χρησιμοποιούνται σε προγράμματα με πολύ μικρούς χρόνους αναμονής
       - σπαταλούν κύκλους CPU χωρίς να απαιτούν τη συνδρομή του ΛΣ
    4. mutexes
       - μία διεργασίας καλεί να μπει στο ΚΤ
       - αναστέλλεται μέχρι να ελευθερωθεί (waiting)
       - ο επεξεργαστής ελευθερώνεται
       - ο χρονοδρομολογητής επιλέγει την επόμενη διεργασία από το ΛΣ
       - η διεργασία αποκτά state: READY
    5. semaphores
       - ένας ακέραιος αριθμός
       - αρχικοποίηση του (αν =1 τότε ισοδυναμούν με mutex)
       - όταν μία διεργασία προσπαθήσει να μπει στο ΚΤ τότε κάνει wait (semaphores -1)
       - αν semaphores >=0 τότε η διεργασία μπαίνει στο ΚΤ
       - αλλιώς συνεχίζει στο wait
       - όταν η διεργασία του ΚΤ τελειώνει τότε καλείται η signal (semaphoresn+1)
       - αν semaphores >=0 τότε μπαίνει η επόμενη της ουράς αναμονής

τα semaphores ελέγχουν την πρόσβαση σε περισσότερα από ένα νήματα- διεργασίες και γίνεται αμοιβαίος αποκλεισμός.

οι εντολές signal() and wait() είναι ατομικές εντολές δηλαδή:

* + - εκτελούνται από το υλικό
    - έχουν αποκλειστικότητα (δηλ. κάνουν lock το ΚΤ)

Τα κλειδώματα (locks) παρέχουν έναν ισχυρό μηχανισμό με τον οποίο μπορούμε να λύσουμε προβλήματα συγχρονισμού. Τι παραπάνω παρέχουν οι παρακολουθητές και οι μεταβλητές συνθήκης σε σχέση με τα κλειδώματα ώστε να παρέχονται και αυτοί οι μηχανισμοί στους προγραμματιστές για την επίλυση προβλημάτων συγχρονισμού;

Οι παρακολουθητές είναι κατασκευές των γλωσσών προγραμματισμού που ενσωματώνουν μεταβλητές, διαδικασίες πρόσβασης και κώδικα αρχικοποίησης σε έναν αφηρημένο τύπο δεδομένων. Μόνο μια διεργασία μπορεί να είναι ενεργή στον παρακολουθητή κάθε φορά (οι υπόλοιπες τοποθετούνται σε ουρά αναμονής ) , αφού η πρόσβαση γίνεται με χρήση των δικών του διαδικασιών πρόσβασης (μεταβλητές συνθήκης cwait , csignal). Με αυτό τον τρόπο υλοποιείται αυτόματα ο αμοιβαίος αποκλεισμός , σε σχέση με άλλα κλειδώματα που πρέπει να υλοποιηθεί συγκεκριμένα για κάθε διεργασία , και επιπλέον η κατασκευή του παρακολουθητή είναι ευκολότερη.

Συστήματα αρχείων (=είναι ο τρόπος με τον οποίο το ΛΣ οργανώνει τα αρχεία στο δίσκο)

1. regular files
2. special files (π.χ. συσκευές, directories -> δείκτες σε άλλα αρχεία)

Κάθε αρχείο έχει τη δικιά του μοναδική ταυτότητα

* η οποία αποθηκεύεται σε μία δομή που ονομάζεται FCB(inode) και περιέχει όλα τα μεταδεδομένα του δηλαδή:
  + Μέγεθος αρχείου
  + Δικαιώματα πρόσβασης
  + Χρήστης, λίστες πρόσβασης
  + Τοποθεσία δεδομένων
  + Ημερομηνίες
* Κάθε FCB προσδιορίζεται από ένα inode id μοναδικό
* Σε ένα directory έχουμε ζεύγη ονομα αρχείου – inode id

Μονοπάτι (path) -> ακολουθία αρχείων

* Όποιο αρχείο ξεκινάει με ‘.’ είναι κρυφό
* Σχετικό path
  + Δε ξεκινάει από το root
  + Για να βρούμε το file το ΛΣ βρίσκει το current directory από το PCB της τρέχουσας διεργασίας και διασχίζει το path ανοίγοντας τα αντίστοιχα directories
* Απόλυτο path (από το root)
  + Για να βρούμε το file πάμε στο root και διασχίζουμε το path ανοίγοντας τα αντίστοιχα directories
* Τι συμβαίνει όταν μεταφέρουμε ένα αρχείο σε άλλο directory
  + Σβήνω την καταχώρηση στο παλιό directory
  + Προσθέτω τη καινούργια καταχώρηση στο νέο directory
  + Δε γίνονται αλλαγές στο μέσο αποθήκευσης (δίσκος)
* Δημιουργία αρχείου
  + Δέσμευση χώρου για τα δεδομένα του και το FCB της
  + Ενημέρωση όλων των δομών του ΛΣ π.χ. ευρετήριο, local πίνακας διεργασίας ανοιχτών αρχείων και global πίνακας ανοιχτών αρχείων

Από την οπτική του χρήστη ένα σύστημα αρχείων προσφέρει:

* Δυνατότητα πρόσβασης σε αρχεία
* Οργάνωση των αρχείων σε καταλόγους
* Δικαιώματα πρόσβασης
* Κλήσεις συστήματος για το χειρισμό αρχείων και καταλόγων

Για να υλοποιηθεί ένα σύστημα αρχείων το ΛΣ χρειάζεται

1. Ένα μέσο αποθήκευσης π.χ. δίσκος
2. Σχεδιασμός και υλοποίηση αποθήκευσης δεδομένων σε αυτό(αρχείων και καταλόγων)
   1. Πραγματικά δεδομένα αρχείων/καταλόγων
   2. Μεταδεδομένα αρχείων
   3. Ευρετήριο αρχείων και καταλόγων
      1. Είναι ένας γρήγορος τρόπος να οργανώνονται τα αρχεία με βάση το inode id για να βρίσκονται γρήγορα (στο δίσκο)
3. Δομές δεδομένων και κώδικας (κλήσεις συστήματος) για το χειρισμό των αρχείων και καταλόγων
   1. Ανοιχτά αρχεία με το FCB τους
   2. Global (όλα τα FCBs) : Ανοιχτά αρχεία ανά διεργασία, τα δικαιώματα τους
   3. Local πίνακας στο PCB κάθε διεργασίας με τα ανοιχτά αρχεία
      1. Για κάθε αρχείο έχει:
         1. Έναν δείκτη στον global πίνακα αρχείων
         2. δικαιώματα
         3. έναν δείκτη που κάνει read, έναν άλλον για write ανάλογα με τα δικαιώματα του αρχείου
            1. έτσι όταν καλώ π.χ. τη read μέσα σε while να μην διαβάζει το file κάθε φορά από την αρχή αλλά να το διαβάζει ολόκληρο προχωρώντας κάθε φορά κατά ένα bit αποθηκεύοντας τη θέση του στο δείκτη -> fd
   4. Τρέχουσα θέση ανά αρχείο – fd (file descriptor)

Open()

* Το ΛΣ βρίσκει το current directory από το PCB της τρέχουσας διεργασίας
* Διαβάζει το inode id του αρχείου που θέλουμε
* Βρίσκει το inode id στο ευρετήριο και παίρνουμε το FCB του αρχείου, δηλαδή όλα τα μεταδεδομένα του
* Βλέπουμε αν έχουμε δικαιώματα πρόσβασης
* Ενημερώνουμε της δομες του ΛΣ (global – local πινακας ανοιχτών αρχείων)
* Επιστροφή file descriptor (δηλαδή επιστρέφουμε τον αριθμό θέσης στο local table)

Read()

* Από το file descriptor βρίσκω τη θέση του αρχείου στο local πίνακα
* Βρίσκω τη θέση του στον global
* Παίρνω το FCB του
* Ανάγνωση δεδομένων (κάνω reαd)
* Ενημερώνω τον δείκτη του read στο local πίνακα για τη διεργασία

Ποια είναι η χρησιμότητα του συστήματος αρχείων (file system) σε ένα Λειτουργικό Σύστημα;

* παρέχει αποδοτική και βολική πρόσβαση στο δίσκο
* επιτρέπει σε δεδομένα να αποθηκεύονται, να εντοπίζονται και να επαναφέρονται εύκολα

Διαφορές soft/hardlinks. Τι συμβαίνει όταν διαγραφεί ένα αρχείο πάνω στο οποίο έχει δημιουργηθεί ένα softlink / hardlink;

Ένα softlink λειτουργεί σαν ένας pointer που δείχνει στο όνομα ενός αρχείου (σαν ένα ζεύγος ονομάτων -> ένα όνομα που δείχνει στο όνομα του αρχείου)  
Το hardlink λειτουργεί σαν ένας pointer που δείχνει στο ίδιο το αρχείο (στο ζεύγος όνομα αρχείου και inode (FCB))

Το ΛΣ αποθηκεύει τον αριθμό hardlinks/softlinks που δείχνουν σε ένα αρχείο (reference counter) για αυτό:

Αν διαγραφτούν όλα τα hardlinks ενός αρχείου τότε διαγράφεται και το ίδιο το αρχείο

Αν διαγραφτούν όλα τα softlinks τότε ΤΙΠΟΤΑ.

Αν διαγραφτεί το αρχείο τότε softlinks -> NULL (εξακολουθούν να υπάρχουν)

Εικονικές Μηχανές

**Μνήμη**: θέλω max αξιοποίηση μνήμης, υψηλή επίδοση, απλότητα

Συνεχόμενη ανάθεση Κύριας Μνήμης

* + Το ΛΣ παραμένει συνεχώς φορτωμένο σε ένα εύρος διευθύνσεων
  + Κάθε νέα διεργασία έχει συγκεκριμένες απαιτήσεις σε μνήμη
  + Οι διεργασίες περιμένουν μέχρι κατάλληλη συνεχόμενη διαμέριση να γίνει διαθέσιμη -> αναμονή
  + Πρόβλημα: εξωτερικός κατακερματισμός -> επίλυση σύμπτυξη (υψηλό κόστος σε χρόνο και υλικό)
  + Απαιτούνται δύο επιπλέον καταχωρητές που ελέγχονται από την MMU και προσδιορίζουν τα επιτρεπτά όρια προσπέλασης μνήμης
    1. Base register
    2. Limit register

Εξωτερικός κατακερματισμός

* + ύπαρξη πολλών μικρών αδέσμευτων περιοχών μνήμης
  + δεν μπορεί να εξυπηρετηθεί αίτηση για δέσμευση μεγάλου ποσού συνεχόμενης μνήμης
  + παρόλο που συνολικά ο διαθέσιμος χώρος επαρκεί

Σελιδοποίηση

* + διαχωρισμός της διαθέσιμης μνήμης σε blocks/σελίδες
  + κάθε σελίδα = page number + offset
    1. offset είναι η μετατόπιση στη σελίδα page\_number[offset]
  + κάθε διεργασία μπορεί να καταλαμβάνει όσο διαθέσιμα pages χρειάζεται
  + Λύνει τον εξωτερικό κατακερματισμό
  + Μειονέκτημα: εσωτερικός κατακερματισμός
  + Το μέγεθος της σελίδας συνηθίζεται να είναι δύναμη του 2 για να πραγματοποιείται εύκολα ο διαχωρισμός αριθμού σελίδας (pagenumber) και μετατόπησης (offset).

Εσωτερικός κατακερματισμός

* + ύπαρξη αχρησιμοποίητων περιοχών μνήμης μέσα σε περιοχές δεσμευμένες από διεργασίες
  + κάθε διεργασία καταλαμβάνει περισσότερο χώρο από ό,τι χρειάζεται γιατί η μνήμη της μπορεί να μην είναι πολλαπλάσιο του μεγέθους της σελίδας

Πίνακας Σελίδων

* + γράφεται από το ΛΣ
  + συμβουλεύεται από αυτό το υλικό(MMU) για αντιστοίχιση
  + περιλαμβάνει μεταφράσεις των λογικών διευθύνσεων σε φυσικές της μνήμης
  + περιλαμβάνει:
    1. αριθμό σελίδας
    2. bit πρόσβασης (r,w,x)
    3. bit εγκυρότητας (valid/invalid)

invalid -> τότε δε βρίσκεται στην κύρια μνήμη αλλά στο δίσκο και πρέπει να γίνει μεταφορά

page fault

η διεργασία πέφτει σε sleep mode και γίνεται context switch με την επόμενη διεργασία του χρονοδρομολογητή στην ουρά αναμονής επειδή η μεταφορά από το δίσκο είναι χρονοβόρα διαδικασία

χάρτης μνήμης

αλλαγή page table με φυσική μνήμη

invalid -> valid

* + 1. modify bit

αν το block έχει αλλαχθεί από τότε που μεταφέρθηκε από το δίσκο οπότε

αν όχι τότε τίποτα

αν ναι τότε θα πρέπει να γίνει replace στο δίσκο με νέο αντίγραφο

* + ΔΕΝ αλλάζει όταν η διεργασία:
    1. Κάνει malloc
    2. Ανοίγει αρχείο
    3. Απεικονίζει ένα αρχείο στη μνήμη

Χάρτης Μνήμης

* + δημιουργείται με τη γέννηση της διεργασίας
  + βρίσκεται στο PCB της
  + επικοινωνεί με το λειτουργικό και μας ενημερώνει κάθε φορά αν περιλαμβάνει όλη τη πληροφορία για τη μνήμη κάθε διεργασίας
    1. πόσες σελίδες έχει και που βρίσκονται
    2. μία σελίδα μπορεί να βρίσκεται στη μνήμη, στο δίσκο ή να μην έχεις το δικαίωμα να τη φορτώσεις
  + τον συμβουλεύεται το ΛΣ αν γίνει page fault
  + ανανεώνεται αν το πρόγραμμα ζητήσει
    1. ζητήσει να μεγαλώσει τη στοίβα του
    2. ελευθερώσει δυναμικά μνήμη στο σωρό (malloc/free)
    3. απεικονίσει ένα αρχείο στη μνήμη

Αν δεν υπάρχει ελεύθερη μνήμη να φέρεις νέα πληροφορία από το δίσκο τότε πρέπει να γίνει replace κάποιο frame. Η επιλογή γίνεται με:

FIFO (faults -> πόσα replaces)

LRU (με βάση το παρελθόν βγάζω αυτή που θα χρειαστεί πιο μετά)

Τι συμβαίνει στο σύστημα Εικονικής Μνήμης και γενικά στο ΛΣ όταν το πρόγραμμα μου εκτελεί:

* κλήση συνάρτησης ή υπορουτίνας
* malloc:
  + Αίτηση δυναμικής μνήμης: αν υπάρχει διαθέσιμος εικονικός χώρος διευθύνσεων για να εκχωρηθεί η μνήμη στο πρόγραμμα.
  + Χορήγηση εικονικής μνήμης
  + Ενημέρωση page table: ανάθεση σελίδων μνήμης στον εικονικό χώρο διευθύνσεων του προγράμματος.
* mmap:
  + ζητάμε από το ΛΣ να μεταφέρει το αρχείο στη μνήμη
  + έτσι μπορούμε να έχουν κατευθείαν πρόσβαση στο αρχείο χωρίς να ζητάμε κάθε φορα από το ΛΣ να εκτελεί διάφορες εντολές
  + ο χάρτης μνήμης ενημερώνεται
  + έχουμε demand paging οπότε το page table θα ενημερωθεί μόνο αν πάμε να γραψουμε σε αυτό
* fork:
  + Κοινή χρήση σελίδων (Copy-on-Write): γονέας- παιδί μοιράζονται τις σελίδες στη φυσική μνήμη μέχρι να απαιτηθεί η εγγραφή σε μια από τις δύο διαδικασίες. Δεν γίνεται ακόμα αντιγραφή
  + Page table -> δικαιώματα μόνο read /execute
  + Όταν κάποιος από τους δύο πάει να γράψει τότε γίνεται page fault
  + Αναλαμβάνει το ΛΣ
  + Χάρτης μνήμης έλεγχος δικαιωμάτων write
  + Γίνεται αντιγραφή στη φυσική μνήμη

Με ποιον τρόπο η σελιδοποίηση κατ' απαίτηση (demand paging) μπορεί να αποτρέψει την εκτέλεση μιας εντολής στην περιοχή των δεδομένων;

* Στην εικονική μνήμη με demand paging , οι σελίδες φορτώνονται μόνο όταν ζητηθούν από τη διεργασία.
* Ο πίνακας σελίδων και ο χάρτης μνήμης , διαθέτουν ειδικές εγγραφές (bit εγκυρότητας-Valid/Invalid και bit πρόσβασης-Read/Write/Execute) που υποδηλώνουν αν η διεργασία έχει δικαίωμα να αναφερθεί σε έναν συγκεκριμένο χώρο μνήμης και αν μπορεί να την μεταβάλει.
* Με αυτό τον τρόπο αποτρέπεται η εκτέλεση μιας εντολής που θα επηρεάσει διευθύνσεις εκτός της δικαιοδοσίας της και θα μορφοποιήσει δεδομένα που είναι μόνο για ανάγνωση.

Πώς επηρεάζεται ο χάρτης μνήμης και ο πίνακας σελίδων μιας διεργασίας που αρχικοποιεί για πρώτη φορά έναν πίνακα δεδομένων που μόλις έχει δεσμεύσει δυναμικά σε ένα σύστημα που υποστηρίζει εικονική μνήμη με σελιδοποίηση κατ' απαίτηση (demand paging);

*Διευκρίνηση: Η διεργασία γράφει για πρώτη φορά έναν πίνακα που αμέσως στην προηγούμενη εντολή έχει κάνει malloc.*

Κατά την κλήση της malloc ο χάρτης μνήμης ανανεώνεται ώστε να περιέχει τις καινούριες μεταφράσεις για τις θέσεις μνήμης που δεσμεύτηκαν. Ο πίνακας σελίδων, λόγω του demand paging, θα μεταβληθεί μόνο εφόσον η διεργασία απαιτήσει κάποια σελίδα η οποία θα φορτωθεί στη μνήμη από τον δίσκο μετά το χειρισμό ενός page fault. Όταν λοιπόν γίνεται αρχικοποίηση του πίνακα, εγγράφεται δηλαδή η πρώτη θέση αυτού , το ΛΣ θα ανανεώσει τον πίνακα σελίδων μόνο με τη σελίδα που εμπεριέχει τη μετάφραση για τη διεύθυνση του πρώτου αυτού στοιχείου.

Πώς επηρεάζεται ο χάρτης μνήμης και ο πίνακας σελίδων μιας διεργασίας που εκτελεί fork() σε ένα σύστημα που υποστηρίζει εικονική μνήμη με σελιδοποίηση κατ' απαίτηση (demand paging) και αντιγραφή κατά την εγγραφή (Copy on Write-CoW);

Όταν η διεργασία-γονέας εκτελέσει fork() θα δημιουργηθεί μια διεργασία-παιδί με κοινή μνήμη και αρχεία με το παιδί αλλά με read-only δικαιώματα πρόσβασης στις σελίδες του γονέα, λόγω της πολιτικής COW . Επίσης, η διεργασία-παιδί αντιγράφει τον χάρτη μνήμης και τον πίνακα σελίδων του γονέα. Όταν αυτή θα θελήσει να μεταβάλλει κάποια από τις σελίδες που διαμοιράζεται με τον γονέα, ΤΟΤΕ αυτή θα αντιγραφεί και ο πίνακας σελίδων και ο χάρτης μνήμης του παιδιού θα ανανεωθούν. Συνεπώς, ο χάρτης μνήμης και ο πίνακας σελίδων της διεργασίας-γονέα δεν επηρεάζεται από τυχόν αλλαγές που πραγματοποιεί η διεργασία-παιδί.

Ποια είναι η χρησιμότητα του χάρτη μνήμης και του πίνακα σελίδων σε ένα σύστημα εικονικής μνήμης με σελιδοποίηση κατ' απαίτηση;

Ο χάρτης μνήμης περιέχει τις μεταφράσεις των σελίδων μιας διεργασίας που βρίσκονται στη μνήμη και στον δίσκο. Ο πίνακας σελίδων είναι υποσύνολο του χάρτη μνήμης και περιέχει μόνο της μεταφράσεις των σελίδων που βρίσκονται στη μνήμη. Έτσι, όταν η διεργασία αναφέρεται σε κάποια σελίδα, γίνεται έλεγχος στον πίνακα σελίδων και αν η σελίδα δεν έχει φορτωθεί στη μνήμη, προσπελαύνεται ο χάρτης μνήμης και ακολουθεί φόρτωση αυτής από τον δίσκο.

Πώς επηρεάζεται ο χάρτης μνήμης και ο πίνακας σελίδων μιας διεργασίας που απεικονίζει ένα αρχείο στη μνήμη (εκτέλεση system call mmap()) σε ένα σύστημα που υποστηρίζει εικονική μνήμη με σελιδοποίηση κατ' απαίτηση (demand paging);

Το memory mapping ενός αρχείου επιτυγχάνεται αντιστοιχίζοντας ένα block δίσκου σε μια σελίδα (ή σελίδες) στη μνήμη , δηλαδή ένα μέρος του χώρου εικονικών διευθύνσεων συσχετίζεται λογικά με το αρχείο. Η αρχική προσπέλαση σε αυτό έχει ως αποτέλεσμα page fault , λόγω demand paging. Αρχικά, ο χάρτης μνήμης ανανεώνεται για να εμπεριέχει την απεικόνιση του αρχείου στον δίσκο. Αντίθετα, ο πίνακας σελίδων θα μεταβληθεί μόνο όταν γίνει αναφορά σε κάποια νέα σελίδα και προκληθεί page fault.

Με ποιον τρόπο η σελιδοποίηση κατ' απαίτηση (demand paging) μπορεί να υποστηρίξει κοινή μνήμη ανάμεσα σε δύο διεργασίες;

Στην εικονική μνήμη με demand paging, οι σελίδες φορτώνονται μόνο όταν ζητηθούν από τη διεργασία. Επιπλέον ο πίνακας σελίδων και ο χάρτης μνήμης , διαθέτουν ειδικές εγγραφές (bit εγκυρότητας-Valid/Invalid και bit πρόσβασης-Read/Write/Execute) που υποδηλώνουν αν η διεργασία έχει δικαίωμα να αναφερθεί σε έναν συγκεκριμένο χώρο μνήμης και αν μπορεί να την μεταβάλει. Έτσι δύο διεργασίες μπορεί να μοιράζονται ένα κοινό χώρο μνήμης ( ειδικά στη περίπτωση της fork() ) με δικαιώματα read-only. Όταν η μία επιχειρήσει να αλλάξει δεδομένα σε κάποια κοινή σελίδα , η σελίδα αυτή θα αντιγραφεί και ο πίνακας σελίδων της διεργασίας θα ανανεωθεί ( τεχνική Copy on Write ).

Γιατί τα σύγχρονα υπολογιστικά συστήματα χρησιμοποιούν πολυεπίπεδη σελιδοποίηση;

Η πολυεπίπεδη σελιδοποίηση τμηματοποιεί τον πίνακα σελίδων (που έχει πολύ μεγάλο μέγεθος), έτσι ώστε να καταλαμβάνουν σημαντικά μικρότερο μέγεθος στη μνήμη. Απαιτούνται, όμως, όπως είναι φυσικό ν-προσβάσεις στη μνήμη όπου ν := πλήθος πινάκων. Αυτό μπορεί να αντιμετωπιστεί με χρήση TLB.

Αναφέρετε ένα πλεονέκτημα και ένα μειονέκτημα της πολυεπίπεδης (ιεραρχικής) σελιδοποίησης έναντι της σελιδοποίησης ενός επιπέδου.

* + Πλεονέκτημα: χρειάζεται μικρότερος χρόνος για την διατήρηση των πινάκων σελίδων
  + Μειονέκτημα: αύξηση του κόστους διάσχισης των πινάκων σελίδων (π.χ. σε περίπτωση TLB miss).

Έστω σύστημα με εικονική μνήμη βασισμένη σε κατ’ απαίτηση σελιδοποίηση (demand paging). Μεταβάλουμε το μέγεθος της σελίδας, ενώ όλα τα άλλα χαρακτηριστικά του συστήματος (υλικό και λογισμικό) παραμένουν σταθερά.

Είναι δυνατόν διπλασιασμός του μεγέθους της σελίδας να οδηγήσει σε μείωση των σφαλμάτων σελίδας (pagefaults);

Ναι. Αν, για παράδειγμα, η διεργασία δεσμεύσει μεγάλο χώρο μνήμης και τον προσπελάσει ακουλουθιακά (όλες τις διευθύνσεις από την αρχή ως το τέλος). Τότε κάθε πρόσβαση σε νέα σελίδα θα δημιουργεί σφάλμα σελίδας. Ο αριθμός των σφαλμάτων είναι: μέγεθος δεσμευμένης μνήμης/μέγεθος σελίδας. Άρα, αύξηση του μεγέθους σελίδας οδηγεί σε μείωση των σφαλμάτων σελίδας.

Είναι δυνατόν μείωση του μεγέθους της σελίδας να οδηγήσει σε μείωση των σφαλμάτων σελίδας (pagefaults);

Ναι. Έστω μία διεργασία που χρησιμοποιεί χώρο μνήμης ίσο με δύο φορές το μέγεθος της φυσικής μνήμης. Έστω, επίσης, ότι κάνει μεγάλο αριθμό προσβάσεων στη μνήμη και ότι όλες οι προσβάσεις της είναι στην αρχη των σελίδων. Τότε θα υπάρχουν σφάλματα σελίδας επειδή δεν χωράνε όλα τα δεδομένα στη μνήμη. Αν μειωθεί το μέγεθος της σελίδας στο μισό, τότε οι προσβάσεις θα γίνονται μόνο στις μισές σελίδες. Σε αυτή τη περίπτωση όλα τα δεδομένα που χρειάζεται η εφαρμογή χωράνε στη μνήμη και κατά συνέπεια μειώνονται τα σφάλματα σελίδας.

Τι εκτυπώνει ο παρακάτω κώδικας αν υποθέσουμε ότι η fork() επιτυγχάνει;

~ int x=0;

~ p = fork(); //p = 0 in child's process p = childs\_PID in father's process

~ if (p > 0) x++; //only father increases the x variable

printf("%d\n", x);

Μετά τη κλήση της fork() το θετικό PID του παιδιού επιστρέφεται στον πατέρα και στο παιδί ανατίθεται μηδενικό PID. Άρα στην συνθήκη για p>0 θα εισέλθει μόνο ο πατέρας ενώ και τα δύο θα εκτελέσουν την printf, δηλαδή θα τυπωθεί είτε 0 \n 1 είτε 1 \n 0 , καθώς απουσία συγχρονισμού δεν μπορούμε να καθορίσουμε ή να γνωρίζουμε τη σειρά εκτέλεσης.

Πόσες διεργασίες δημιουργεί ο κώδικας: for (i=0; i<3; i++) fork(); 8