Λειτουργικά Συστήματα – Εργασία

ΑΝΑΦΟΡΑ ΠΑΡΑΔΟΣΗΣ

(**Όνομα:** Φοίβος Παπαθανασίου, **ΑΜ:** 3200138, **Email:** p3200138@aueb.gr)

Σημείωση:

Η εξυπηρέτηση γίνεται με πολιτική **FCFS** (First-Come First-Serve), δηλαδή προτεραιότητα έχει ο πελάτης που έκανε πιο νωρίς την παραγγελία έτσι ώστε να επιτυγχάνονται ικανοποιητικοί χρόνοι εξυπηρέτησης ενώ βρισκόμαστε και ύπο τους περιορισμούς της εκφώνησης της εργασίας.

Αυτό σημαίνει ότι εάν για παράδειγμα υπάρχουν αρκετοί φούρνοι για την παραγγελία με oid=3 αλλά δεν υπάρχουν αρκετοί φούρνοι για την παραγγελία με oid=2, η παραγγελία με oid=3 πρέπει να περιμένει ώστε να δεσμεύσει η παραγγελία με oid=2 αρκετούς φούρνους. (Σημειώνεται ότι οι πίτσες πρέπει να ψήνονται παράλληλα όλες μαζί οπότε δε γίνεται να εξυπηρετηθεί η oid=3 μέχρι να βρει αρκετούς φούρνους η oid=2 γιατί μπορεί να στερηθεί το δικαίωμα της προτεραιότητας)

Συναρτήσεις:

void initializeMutex(pthread_mutex_t *mutex): Χρησιμοποιείται μόνο στη main. Αρχικοποιεί mutex, σε περίπτωση αποτυχίας εκτυπώνει κατάλληλο μύνημα και το πρόγραμμα τερματίζεται με code -1.

void initializeCondition(pthread_cond_t *cond): Χρησιμοποιείται μόνο στη main. Αρχικοποιεί mutex, σε περίπτωση αποτυχίας εκτυπώνει κατάλληλο μύνημα και το πρόγραμμα τερματίζεται με code -1.

void acquireLock(pthread_mutex_t *mutex, int oid, void* t): Επιχειρεί κλείδωμα. Αν αποτύχει, τότε εκτυπώνεται το <oid> και το πρόγραμμα τερματίζεται.

void releaseLock(pthread_mutex_t *mutex, int oid, void* t): Απελευθερώνει το κλείδωμα. Αν η απελευθέρωση αποτύχει, τότε το εκτυπώνεται το <oid> και το πρόγραμμα τερματίζεται.

void destroyLock(pthread_mutex_t *mutex): Καταστρέφει mutex. Εάν αποτύχει, εκτυπώνει κατάλληλο μύνημα και το πρόγραμμα τερματίζεται με code -1.

void destroyCond(pthread_cond_t *cond): Καταστρέφει cond. Εάν αποτύχει, εκτυπώνει κατάλληλο μύνημα και το πρόγραμμα τερματίζεται με code -1.

int bernoulliDistr(float p, void* arg): Επιστρέφει 1 με πιθανότητα p και 0 με πιθανότητα 1-p. Ως δεύτερη παράμετρο περνάμε τη διεύθυνση ενός seed για την rand_r().

int main(int argc, char* argv[]): Αρχικά αρχικοποιούμε όλα τα locks και conds. Στη συνέχεια, ελέγχουμε εάν έχουν δοθεί οι δύο παράμετροι (Ncustomers, initial seed). Ακολούθως δεσμεύουμε μνήμη για τα threads και χρησιμοποιώντας την pthread_create( ) κατασκευάζουμε τα threads που αντιστοιχούν στους πελάτες. Ως argument στη συνάρτηση των threads περνάμε το oid που αντιστοιχεί στο αναγνωριστικό παραγγελίας του πελάτη, επίσης κάθε φορά - με εξαίρεση την πρώτη δημιουργία νήματος - χρησιμοποιούμε τη συνάρτηση sleep( ) για [Torderlow, Torderhigh] ώστε να προσομοιώσουμε το χρόνο που κάνει για να έρθει νέος πελάτης. Μετά, η main( ) περιμένει με τη μέθοδο pthread_join( ) να ολοκληρώσουν τα νήματα και στη συνέχεια εάν όλα πάνε καλά εκτυπώνει τα απαιτούμενα στατιστικά. Τέλος, καταστρέφονται τα locks, conds και απελευθερώνεται η δεσμευμένη μνήμη.

void *simulateServiceFunc(void *t):

Τις λειτουργίες της συνάρτησης αυτής μπορούμε να τις χωρίσουμε χονδρικά σε 7+1 μέρη:

Αρχικοποίηση

Στο σημείο αυτό δηλώνουμε και αρχικοποιούμε ορισμένες μεταβλητές που θα χρειαστούμε. Σημαντικό είναι το explicit casting της μεταβλητής t ώστε να λάβουμε το argument που πέρασε η pthread_create( ). Επίσης αρχικοποιούμε το seed σε κάθε νήμα ώς εξής: seed = initialSeed + *oid έτσι ώστε να μπορούμε να χρησιμοποίησουμε ορθά την rand_r( ).

Μέρος 1ο

Στο πρώτο μέρος καθορίζουμε το πλήθος των συνολικών πιτσών, των special πιτσών και των κανονικών πιτσών. Για να το επιτύχουμε αυτό κάνουμε χρήση της συνάρτησης bernoulliDistr( ) και ώς πρώτη παράμετρο περνάμε το Pplain(πιθανότητα η πίτσα να είναι απλή).

Μέρος 2ο

Στο μέρος αυτό προσομοιώνουμε τον χρόνο συναλλαγής ([Tpaymentlow,Tpaymenthigh]) με χρήση της μεθόδου sleep( ). Στη συνέχεια, ελέγχουμε εάν η συναλλαγή αποτυγχάνει η όχι και ενημερώνουμε ανάλογα τις μεταβλητές failedTransactionCount, succesfulTransactionCount, totalPlainSold, totalSpecialSold, totalIncome. Για να το επιτύχουμε αυτό με thread-safe τρόπο, χρησιμοποιούμε δύο locks, το πρώτο για το κλείδωμα

οθόνης (outputLock) και το δεύτερο (transactionLock) για να μπορέσουμε να επηρεάσουμε με ασφάλεια τις μεταβλητές που προαναφέρθηκαν. Σημειώνεται ότι στην περίπτωση αποτυχίας συναλλαγής το νήμα απελευθερώνει όλα τα locks και τερματίζει με χρήση της μεθόδου pthread_exit( ). Τέλος, στην περίπτωση επιτυχούς συναλλαγής ανατίθεται ο αριθμός προτεραιότητας (orderPriority) του πελάτη (νήματος) ανάλογα με τη σειρά που πραγματοποίησε την παραγγελία ώστε να επιβληθεί η πολιτική FCFS στη συνέχεια.

Μέρος 3ο

Με χρήση των cookPriorityCond,CookPriorityLock και cookPriority εξασφαλίζουμε ότι το νήμα με το μικρότερο oid έχει προτεραιότητα στη δέσμευση παρασκευαστή. Ακολούθως με χρήση των availableCookLock, availableCookCond και availableCookCount εξασφαλίζουμε ότι κάθε νήμα θα δεσμεύει παρασκευαστή με thread-safe τρόπο.

Αναλυτικά, όταν αποκτήσει lock στην availableCookLock ένα νήμα, συνεχίζει σε ένα while loop (για αποφυγή spurious wakeups) όπου ελέγχεται εάν υπάρχει ελεύθερος παρασκευαστής.

Εάν όχι τότε το νήμα περιμένει ειδοποίηση από άλλο νήμα ότι υπάρχει διαθέσιμος παρασκευαστής, εάν ναι τότε μειώνει την availableCookCount κατά 1 και απελευθερώνει το availableCookLock, στη συνέχεια αφού πλέον έχει αποκτήσει παρασκευαστή, αυξάνει την cookPriority κατά 1, απελευθερώνει το cookPriorityLock και ενημερώνει τα νήματα που περιμένουν τη σειρά τους για να επιχειρήσουν να δεσμεύσουν παρασκευστή (βασικά ένα νήμα μόνο θα έχει το δικαίωμα και είναι αυτό με το μικρότερο oid από τα άλλα, παρόλαυτα αναγκαστικά θέλουμε ένα pthread_cond_broadcast( ) γιατί πολλά νήματα μπορεί να περιμένουν εκεί).

Τέλος προσομοιώνουμε τον χρόνο παρασκευής των πιτσών.

Μέρος 4ο

Ομοίως με το μέρος 3ο επιβάλλουμε πολιτική FCFS για τους φούρνους τώρα με χρήση των ovenPriorityLock, ovenPriorityCond και ovenPriority.

Η thread-safe δέσμευση των φούρνων επιτυγχάνεται με την χρήση των availableOvenLock, availableOvenCond και availableOvenCount με ανάλογο τρόπο που πραγματοποιείται και στο μέρος 3. Η παραλλαγή είναι στο γεγονός ότι αυτή τη φορά το while loop είναι απαραίτητο όχι μόνο για τα spurious wakeups αλλά και γιατί κάθε παραγγελία απαιτεί διαφορετικό αριθμό φούρνων συνεπώς πρέπει να ελέγχουμε εάν έχει αποδεσμευθεί ο απαραίτητος αριθμός φούρνων ώστε να μπορέσουμε να προχωρήσουμε στο ψήσιμο των πιτσών, δηλαδή μπορεί να αποδεσμευθούν μερικοί φούρνοι, να ξυπνήσει το νήμα όμως να μην επαρκεί το πλήθος των διαθέσιμων φούρνων για να ψηθούν **όλες οι πίτσες παράλληλα** και συνεπώς να πρέπει να ξαναπεριμένει.

Ακολούθως, αφού δεσμευτούν οι φούρνοι και επιτρέψουμε στα άλλα νήματα να διεκδηκήσουν φούρνους, απελευθερώνουμε τον παρασκευαστή χρησιμοποιώντας το availableCookLock και ενημερώνουμε με τη μέθοδο pthread_cond_signal το νήμα που περιμένει για φούρνους να ξυπνήσει. (θα είναι το πολύ ένα κάθε δεδομένη χρονική στιγμή λόγω της πολιτικής FCFS).

Τέλος, προσομοιώνουμε τον χρόνο ψησίματος των πιτσών (Tbake) και κρατάμε και την ώρα ολοκλήρωσης ψησίματος με χρήση της μεθόδου clock_gettime( ) ώστε να μπορέσουμε να υπολογίσουμε τους χρόνους κρυώματος αργότερα.

Σημείωση: Ως πρώτη παράμετρο στην clock_gettime περνάμε την CLOCK_REALTIME όπου της λέει στην ουσία να επιστρέψει την ώρα που λέει ο υπολογιστής μας, και τη διεύθυνση ενός timespec struct ώστε να αποθηκευτούν σε αυτό τα sec και ns που αθροιστικά αντιπροσωπεύουν την ώρα.

Μέρος 5ο

Στο μέρος αυτό επιβάλλεται η πολιτική FCFS για τους υπάλληλους πακεταρίσματος με τη χρήση των packerPriorityLock, packerPriorityCond και packerPriority.

Παρομοίως με τα μέρη 3 και 4, όταν έρθει η σειρά του νήματος να δεσμεύση υπάλληλο πακεταρίσματος, αυτό επιτυγχάνεται με thread-safe τρόπο χρησιμοποιώντας τα availablePackerLock, availablePackerCond και availablePackerCount.

Ακολούθως, αφού δεσμευτεί υπάλληλος πακεταρίσματος, προσομοιώνουμε τον χρόνο πακεταρίσματος (Tpack * πλήθος πιτσών παραγγελίας) με τη sleep( ). Στη συνέχεια, καταγράφουμε την ώρα ολοκλήρωσης του πακεταρίσματος με χρήση της μεθόδου clock_gettime( ) ώστε στη συνέχεια να τη χρησιμοποιήσουμε για να υπολογίσουμε το χρόνο ετοιμασίας της παραγγελίας (χρονική στιγμή που μπήκε ο πελάτης εώς την χρονική στιγμή όπου παρασκευάστηκε η παραγγελία).

Στη συνέχεια μπορούμε να αποδεσμεύσουμε τους φούρνους οπότε κάνουμε acquire το availableOvenLock, ενημερώνουμε τη μεταβλητή availableOvenCount και ενημερώνουμε το νήμα που περιμένει με τη μέθοδο pthread_cond_signal( ), τέλος απελευθερώνουμε το lock.

Ομοίως αποδεσμεύουμε και τον υπάλληλο πακεταρίσματος.

Τέλος, υπολογίζουμε το χρόνο προετοιμασίας της παραγγελίας σε λεπτά και χρησιμοποιώντας κάνωντας acquire lock στην outputLock εκτυπώνουμε ενημερωτικό μύνημα σχετικά με το χρόνο προετοιμασίας της παραγγελίας με το εκάστοτε <oid>, στη συνέχεια απελευθερώνουμε το lock.

Μέρος 6ο

Καταρχάς επιβάλλουμε την πολιτική FCFS για τον ντελιβερα χρησιμοποιώντας τις μεταβλητές delivererPriorityLock, delivererPriorityCond και delivererPriority.

Όταν έρθει η σειρά του νήματος να δεσμεύσει ντελιβερά, αρχικά κάνει acquire lock στην availableDelivererLock για ασφαλή πρόσβαση στη μεταβλητή availableDelivererCount, στη συνέχεια μέσα σε ένα while loop (για αποφυγή spurious wakeups) ελέγχει εάν υπάρχει διαθέσιμος ντελιβεράς, εάν δεν υπάρχει, τότε το νήμα κοιμάται μέχρι να αποδεσμευθεί κάποιος ντελιβεράς. Εάν υπάρχει, τότε δεσμεύει έναν ντελιβερά (availableDeliverer Count -= 1) και απελευθερώνει το lock στην availableDeliverer Lock. Στη συνέχεια, αυξάνει τη μεταβλητή deliverer Priority κατά ένα ώστε να μπορέσει να δεσμεύσει ντελιβερά το νήμα με το επόμενο μικρότερο <oid> και απελευθερώνει το lock στην deliverer PriorityLock, τέλος με χρήση της pthread_cond_broadcast( ) μέσω της deliverer PriorirtCond ενημερώνει τα νήματα που περιμένουν τη σειρά τους να ξυπνήσουν.

Ακολούθως, προσομοιώνουμε τον χρόνο μέχρι να φτάσει ο ντελιβεράς στον πελάτη ([Tdellow,Tdelhigh]) με χρήση της sleep( ) και μετά παίρνουμε την ώρα όπου έφτασε με τη μέθοδο clock_gettime( ) ώστε να υπολογίσουμε το χρόνο κρυώματος παραγγελίας καθώς και το χρόνο ολοκλήρωσης παραγγελίας.

Έπειτα, υπολογίζουμε τον χρόνο ολοκλήρωσης της παραγγελίας και κάνωντας acquire lock στην outputLock εκτυπώνουμε σχετικό μήνυμα ενημέρωσης. Αποδεσμεύουμε το lock στην outputLock.

Προσομοιώνουμε το χρόνο μέχρι να επιστρέψει ο ντελιβεράς.

Απελευθερώνουμε τον ντελιβερά χρησιμοποιώντας τις availableDeliverer Lock και availableDeliverer Count και ενημερώνουμε το νήμα που περιμένει με τη pthread_cond_signal( ) μέσω της availableDeliverer Cond.

Μέρος 7ο

Υπολογίζουμε το coolingTime σε λεπτά, ακολούθως κάνουμε acquire lock στη μεταβλητή statsLock που θα μας επιτρέψει με ασφαλή τρόπο να χρησιμοποιήσουμε τις μεταβλητές maxOrderCompletionTime, maxCoolingTime, orderCompletionTimeSum και coolingTimeSum ώστε στο τέλος στη main( ) να εκτυπώσουμε τα απαιτούμενα στατιστικά.

Τέλος απελευθερώνουμε το lock στην statsLock και τερματίζουμε το νήμα με την μέθοδο pthread_exit( ).

Βοηθητικό Διάγραμμα: