Λειτουργικά Συστήματα
Εργασία 3
Τεκμηρίωση Κώδικα

Λάμπρου Ιωάννης

1 Οκτωβρίου 2017

Στο παραδοτέο αρχείο περιέχονται τα αρχεία: main.c, exp_rand.cpp, exp_rand.h, process.c, process.h, dl_node.cpp, dl_node.h, process_queue.cpp, process_queue.h, priority_queue.cpp, priority_queue.h, sem_queue.cpp, sem_queue.h, Makefile. Στο main.c περιέχεται ο κύριος κώδικας της προσομοιωσης, όπως ζητήθηκε στην εκφώνηση (main simulation loop). Στο αρχείο exp_rand.cpp περιέχεται η συνάρτηση int exp_rand(double avg), η οποία επιστρέφει μια τυχαία ακέραια τιμή (στρογγυλοποιημένη) σε εκθετική κατανομή, παίρνοντας ως είσοδο τον μέσο όρο της avg (εκθετικά κατανεμημένη τιμή). Τα υπόλοιπα αρχεία του παραδοτέου περιέχουν όλες τις διαφορετικές δομές που χρησιμοποιούνται για την προσομοίωση δομών και λειτουργιών του λειτουργικού συστήματος. Τέλος υπάρχει αρχείο makefile με όνομα Makefile.

Η μονάδα μέτρησης του χρόνου είναι το 1 timeslot, ενώ ο αριθμός της πιθανότητας input για είσοδο μιας διεργασίας σε κρίσιμη περιοχή είναι % (πχ 50%)

Για τις ανάγκες της εργασίας, χρησιμοποιήθηκαν οι παρακάτω δομές (κλάσεις):

```
class Process {
private:
    int pid;
    int priority;
    int rem_life;
    // If using_semnum is -1, then it
    // is not using a critical section
    int using_semnum;
    int rem_crit_usg;
    int tot_waited;
    int waiting_since;
public:
    Process(int, int, int);
    ~Process();

    int get_pid();
    int get_priority();
    int get_rem_life();
    int get_using_semnum();
    int get_rem_crit_usg();
    int get_tot_waited();
```

```
    void set_using_semnum(int);
    void set_rem_crit_usg(int);
    void set_waiting_since(int);
    void calc_total_waited(int);
    void execute(int);
};
```

Η κλάση Process χρησιμοποιείται για να αναπαρασταθεί μια διεργασία μέσα στην προσομοίωση. Σε ένα αντικείμενο Process αποθηκεύονται διάφορα χρήσιμα δεδομένα όπως το pid, η προτεραιότητα (η οποία παραμένει πάντα η ίδια, ανεξάρτητα με προσωρινές αλλαγές από σεμαφόρους), η εναπομείνουσα ζωή της διεργασίας, καθώς και αν χρησιμοποιεί κρίσιμη περιοχή (down κάποιον σεμαφόρο και ποιόν), για πόσο ακόμα θα την χρησιμοποιεί και συνολικά πόσο χρόνο έχει περιμένει.

```
class DlNode {
private:
        Process* process_ptr;
        DlNode* nextnode_ptr;
        DlNode* prevnode_ptr;
public:
        DlNode(Process*);
        ~DlNode();

        void set_nextnode(DlNode*);
        void set_prevnode(DlNode*);

        Process* get_process();
        DlNode* get_nextnode();
        DlNode* get_prevnode();
};
```

Η κλάση DlNode (Double linked Node) είναι ο κύριος τύπος κόμβου που χρησιμοποιείται για την αποθήκευση μιας διεργασίας (περιέχει δείκτη σε Process) στις διάφορες δομές δεδομένων του συστήματος το οποίο προσομοιώνεται (PriorityQueue, SemQueue). Αξίζει να σημειωθεί πως, αν και ο κόμβος είναι διπλά συνδεδεμένος, (δείχνει σε επόμενο και προηγούμενο κόμβο) μέσα στο SemQueue χρησιμοποιείται ως απλά, με τον pointer στον προηγούμενο κόμβο να είναι πάντα nullptr.

```
class ProcessQueue {
private:
        // First node of the queue
        DlNode* firstnode_ptr;
```

```
        // Current node of the queue (Process to use current time slot)
        DlNode* currnode_ptr;
        // Node after the current node in the queue
        DlNode* afternode_ptr;

        // Simulation Data
        int total_proc;
        int timeslots_waited;

        void del_all_nodes(DlNode*);
public:
        ProcessQueue();
        ~ProcessQueue();

        // Inserts the input Node before
        // the current node, at the end of the queue
        void insert_node(DlNode*);

        DlNode* get_currnode();
        DlNode* get_firstnode();
        DlNode* extract_currnode();
        DlNode* extract_node_wpid(int);

        void update_data(int);
        int get_total_proc();
        int get_timeslots_waited();
        void find_next_node();
};
```

Η κλάση ProcessQueue χρησιμοποιείται για να αναπαρασταθεί μια κυκλική ουρα ουρά διεργασιών του λειτουργικού συστήματος. Σε κάθε ουρά περιέχονται διεργασίες ίδιας προτεραιότητας και υπάρχει μια ProcessQueue για κάθε δυνατή προτεραιότητα. Ο currnode_ptr δείχνει τον κόμβο ο οποίος είναι πρώτος στην ουρά. Ο afternode_ptr δείχνει τον κόμβο μετά από τον πρώτο (δεξια, next). Όταν εισάγεται ένας νέος κόμβος τοποθετείται στο τέλος της ουράς (πριν από τον πρώτο, αριστερα, prev). Με τη χρήση της void find_next_node() γίνεται η εναλλαγή του πρώτου κόμβο σε τελευταίο, του τελευταίου σε προτελευταίο και του δεύτερου σε πρώτο. Επίσης μπορεί να εξάγει κόμβους για την προσωρινή αλλαγή της προτεραιότητάς τους, ή τη μεταφορά τους σε ουρά σεμαφόρου όταν γίνονται block από αυτόν. Τέλος, σε κάθε ουρά αποθηκεύονται και χρήσιμα δεδομένα για τα αποτελέσματα της προσομοίωσης, όπως ο συνολικός αριθμός διεργασιών με την προτεραιότητα της ουράς ή ο συνολικός αριθμός των χρονοθυρίδων που έχουν αυτές περιμένει.

```
class PriorityQueue {
```

```
private:
        // First node of the queue
        ProcessQueue proc_queues[7];
        // The queue number where the current node is
        int from_queue;
public:
        PriorityQueue();
        ~PriorityQueue();

        // Makes a new DlNode and insets it before
        // the current node, at the end of the queue
        void insert_process(Process*);
        void insert_node_norm(DlNode*);
        void insert_node_wprio(DlNode*, int);
        void update_data(int, int);

        DlNode* get_currnode();
        DlNode* extract_currnode();
        DlNode* extract_node_wprio(int, int);
        // Extracts process, deleting current node
        Process* extract_process();

        void find_next_node();
        bool is_empty();

        void print_data();
};
```

Η κλάση PriorityQueue χρησιμοποιείται για να αναπαρασταθεί η ζητούμενη υλοποίηση μιας ουράς προτεραιότητας διεργασιών του λειτουργικού συστήματος. Περιέχει 7 ProcessQueues, μία για κάθε προτεραιότητα. Είναι η μόνη δομή του προγράμματος η οποία δημιουργεί και καταστρέφει αντικείμενα DlNode κατά την εισαγωγή και εξαγωγή μιας διεργασίας από αυτή (insert_process(), extract_process()). Ακόμη, μπορούν να εισαχθούν κόμβοι κανονικά ή σε μια ουρά με επιλεγμένη προτεραιότητα, ανεξάρτητα από την προτεραιότητα της διεργασίας του κόμβου που εισάγεται. Κάθε φορά που καλείται η void find_next_node() βρίσκεται η επόμενη σε σειρα (after_node) διεργασία της ουράς με την υψηλότερη δυνατή προτεραιότητα και αποθηκεύεται ο αριθμός της ουράς αυτής. Τέλος, υπάρχει η δυνατότητα να εξάγει κόμβους, (τον current, ή με συγκεκριμένη προτεραιότητα και pid) για την προσωρινή αλλαγή της προτεραιότητάς τους, ή τη μεταφορά τους σε ουρά σεμαφόρου όταν γίνονται block από αυτόν.

```
class SemQueue {
private:
        // First node of the queue
```

```
        DlNode* firstnode_ptr;
        // Last node of the queue
        DlNode* lastnode_ptr;
        // Node not in the queue, containing the process
        // that is currently using the critical section
        int using_crit_pid;
        int normal_prio;
        int highest_prio;

        int blocked_by_lower;
        // For safety
        void del_all_nodes(DlNode*);
        // Used by up() and down()
        // Only control the normal queue they do not
        // change the value of using_crit_ptr
        void push(DlNode*);
        DlNode* pop();
public:
        SemQueue();
        ~SemQueue();

        int get_blocked_by_lower();
        void update_data();

        int down(PriorityQueue*, int);
        int up(PriorityQueue*, int);
};
```

Η κλάση SemQueue χρησιμοποιείται για να αναπαρασταθεί η ζητούμενη υλοποίηση μιας ουράς σεμαφόρου του λειτουργικού συστήματος. Δημιουργείται μία ουρά για κάθε σεμαφόρο της προσομοίωσης. Μέσα σε κάθε ουρά τοποθετούνται κόμβοι οι οποίοι περιέχουν διεργασίες οι οποίες έχουν επιχειρήσει να πάρουν πρόσβαση σε κρίσιμη περιοχή και κάνοντας down() τον συγκεκριμένο σεμαφόρο. Στις μεταβλητές using_crit_pid και normal_prio αποθηκεύονται οι τιμές των pid και priority της διεργασίας η οποία κατάφερε επιτυχώς να κάνει down() τον σεμαφόρο (χωρίς να μπλοκαριστεί), αντίστοιχα. Οι μέθοδοι down() και up() δέχονται ως όρισμα έναν δείκτη στο PriorityQueue του συστήματος (προσομοίωσης), έτσι ώστε να μπορεί να μετακινεί κόμβους διεργασιών από μια ProcessQueue μιας προτεραιότητας σε μια άλλη. Τέλος σε κάθε SemQueue αποθηκεύονται δεδομένα χρήσιμα για την προσομοίωση, όπως ο συνολικός αριθμός των χρονοθυρίδων όπου μια διεργασία μπλοκάρεται από μια άλλη διεργασία χαμηλότερης προτεραιότητας, σε αυτήν την ουρά σεμαφόρου.

Η ανάπτυξη της εργασίας έγινε σε περιβάλλον Windows 10 σε γλώσσα C++.

Η εργασία δοκιμάστηκε σε περιβάλλοντα Windows 10 και Ubuntu 16.04.