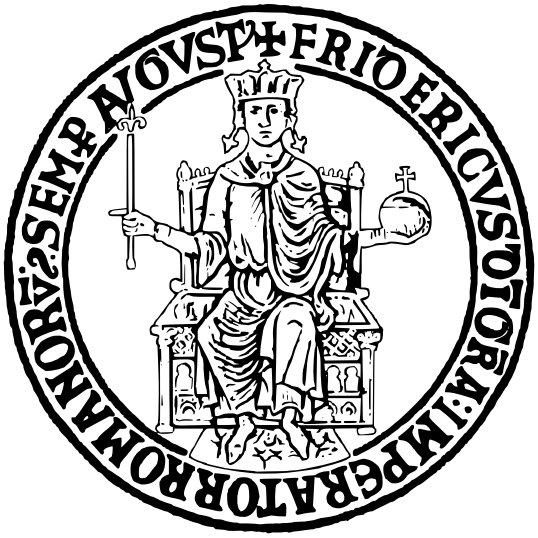

Università degli studi di Napoli Federico II

Supermarket Simulation

Francesco Terrecuso
N86004191

Lorenzo Tecchia
N86004446

Simone Parente Martone
N86004297

Department of Computer Science

November 14, 2024

1 Introduzione

Questo è un documento di presentazione per un progetto sviluppato per l'esame di Laboratorio di Sistemi Operativi, A.A 2023/2024, tenuto dalla Prof. Rossi Alessandra.
Il progetto consiste nella simulazione di un supermercato, modellando le interazioni tra quest'ultimo, i clienti e le casse in un contesto multi-threaded. Questo file si divide in diverse sezioni che forniscono una panoramica dei requisiti, delle scelte architetturali e implementative adottate.

2 Requisiti identificati

Il progetto prevede la realizzazione di un architettura client-server che simuli un supermercato dotato di K casse e avente un limite di C clienti presenti contemporaneamente all'interno.
All'inizio della simulazione, C clienti entrano nel supermercato simultaneamente, successivamente, ogni volta che escono E clienti, possono entrarne altri E.
Ogni cliente ha un tempo variabile dedicato agli acquisti, una volta scaduto questo tempo, il cliente si mette in fila per pagare, attendendo il proprio turno. Ultimato il pagamento, il cliente esce dal supermercato.
Ogni cassa è gestita da un cassiere che serve i clienti in base a una politica FIFO, il tempo di servizio è dato da:

- una componente costante, specifica per ogni cassiere
- una componente variabile, linearmente dipendente dal numero di acquisti

3 Scelte architetturali

L'architettura del sistema si basa sul modello client-server.
Sia il client che il server saranno sviluppati in C. Il server gestisce il funzionamento del supermercato e delle casse, mentre i client simulano il comportamento dei clienti: entrano, fanno acquisti e una volta effettuato il pagamento, escono dal supermercato.
La gestione del supermercato sarà implementata tramite l'uso di thread multipli per garantire una simulazione fluida e ottimizzata delle operazioni. Prevediamo l'utilizzo di:

- Un thread per la supervisione del supermercato.
- Un thread per ogni cassa.
- Un thread per ogni cliente all'interno del supermercato.

La scelta di utilizzare thread invece di processi è stata presa per evitare di gravare eccessivamente sul sistema, minimizzando l'uso delle risorse e garantendo migliori prestazioni.
L'accesso esclusivo al supermercato e alle casse sarà garantito dall'utilizzo di mutex, che assicureranno la sincronizzazione tra i thread, evitando race conditions e conflitti.
La scelta della cassa non è svolta autonomamente dal cliente, bensì dal supermercato: se la cassa è vuota, il cliente viene assegnato a questa cassa, altrimenti viene assegnato alla cassa con meno clienti in coda.

4 Componenti della simulazione

4.1 Clienti

I clienti vengono creati e ammessi al supermercato fino ad un numero prestabilito passato come parametro in fase di esecuzione, i clienti che proveranno ad accedere successivamente saranno messi in una lista d'attesa.
Ogni cliente ha il seguente workflow:

- **Fase di acquisto**: i clienti passano un tempo generato casualmente tra 0 e 190 secondi in questa fase, scegliendo un numero di articoli compreso tra 0 e 19.
- **Coda alle casse**: il cliente rimane in attesa.
- **Pagamento**: il cliente viene servito e attende un tempo variabile.
- **Uscita dal supermercato**: ultimato il pagamento, il cliente esce dal supermercato, chiudendo la connessione.

Ai clienti viene affidato un ID, che non viene generato casualmente, potenzialmente incorrendo in conflitti tra l'associazione univoca di clienti ed ID. Si è risolto tale problema attraverso l'associazione del file descriptor delle socket al id del client. Essendo le socket uniche per ogni cliente che prova ad accedere al supermercato, si mantiene l'unicità dell'id cliente.

4.2 Casse

Il numero di casse è definito all'avvio del server, ogni cassa opera in maniera indipendente, con una propria coda e parametri generati casualmente. Il tempo totale di servizio è calcolato tramite un tempo fisso sommato al prodotto tra un modificatore casuale e il numero di oggetti del cliente. Una volta terminato il servizio di un cliente, la cassa serve il cliente successivo, in caso la coda fosse vuota la cassa rimarrebbe inattiva ed in attesa di altri clienti.

4.3 Supermercato

Il supermercato viene inizializzato all'avvio del server, il numero di casse e di clienti che possono trovarsi all'interno simultaneamente sono valori dati in fase di esecuzione. È dotato di una coda esterna, in cui i clienti attendono che il numero di clienti all'interno scenda sotto una certa soglia prima di poter entrare. Il supermercato si occupa di gestire la scelta della cassa a cui assegnare il clientee lo shift dei clienti in avanti a quella determinata cassa.

5 Funzionalità

Vengono qui riportate tre delle funzioni principali della simulazione del Supermarket. In particolare la funzioni di:

- Servi Cliente: Accede alla struttura dati cassa, mettendo i clienti in fila
- Supervisiona Supermercato: Accede alla struttura dati supermercato, verificando il numero di clienti presenti all'interno del supermercato
- Client Handler: Associa l'id univoco ad ogni cliente e inizializza i parametri di sumulazione per ogni cliente, quali numero di oggetti da comprare, tempo da passare nel supermercato prima di mettersi in fila.

5.1 Servi Cliente

Listing 1: Servi Cliente

```
1  void *servi_cliente(void *arg) {
2      ParametriCassa *parametri = (ParametriCassa *)arg;
3      Cassa *cassa = (Cassa *)parametri->cassa;
4      Supermercato *supermercato = (Supermercato *)parametri->
       supermercato;
5
6      while (1) {
7          srand(time(NULL));
8          pthread_mutex_lock(&cassa->mutex_cassa);
9
10         // Aspetta che ci siano clienti in coda
11         while (cassa->num_clienti_in_coda == 0) {
12             pthread_cond_wait(&cassa->coda_vuota, &cassa->
       mutex_cassa);
13         }
14
15         // Prende il primo cliente in coda (FIFO)
16         Cliente * cliente = &(cassa->coda[0]);
17
18         // Sposta gli altri clienti avanti nella coda
```

```
19          for (int i = 0; i < cassa->num_clienti_in_coda - 1; i
    ++) {
20              cassa->coda[i] = cassa->coda[i + 1];
21          }
22          cassa->num_clienti_in_coda--;
23
24          // Sblocca il mutex per permettere ad altri di
    aggiungere clienti alla coda
25          pthread_mutex_unlock(&cassa->mutex_cassa);
26
27          int tempo_servizio = cassa->tempo_fisso + cliente->
    numero_di_oggetti * rand() % 3 + 1;
28          printf("Cassa %d sta servendo il cliente %d per %d
    secondi [...] \n", cassa->id, cliente->id, tempo_servizio)
    ;
29
30          // Simula il servizio del cliente
31          sleep(tempo_servizio);
32
33          // Aggiorna il tempo totale di servizio della cassa
34          pthread_mutex_lock(&cassa->mutex_cassa);
35          cassa->tempo_totale += tempo_servizio;
36          pthread_mutex_unlock(&cassa->mutex_cassa);
37
38          printf("Cassa %d ha terminato di servire il cliente %
    d.\n", cassa->id, cliente->id);
39
40          //elimina il cliente dalla cassa
41          pthread_mutex_lock(&supermercato->mutex_supermercato)
    ;
42          supermercato->clienti_presenti--;
43          printf("Cliente %d servito e uscito dal supermercato\
    n", cliente->id);
44          pthread_mutex_unlock(&supermercato->
    mutex_supermercato);
45
46      }
47      return NULL;
```

5.2 Supervisiona Supermercato

Listing 2: Supervisiona Supermercato

```
1  void* supervisiona_supermercato(void *arg) {
2      sleep(10);  // Attende 10 secondi prima di iniziare
3
4      Supermercato *supermercato = (Supermercato *)arg;
5      //stampa la lista di attesa con gli id dei clienti
6      printf(COLOR_GREEN "Lista attesa finale di %d Clienti"
```

```
    COLOR_RESET, supermercato->clienti_fuori);
    printf("\n");

    while (1) {
        // Blocca il mutex per leggere e modificare i dati del supermercato
        pthread_mutex_lock(&supermercato->mutex_supermercato);

        // Controlla se possiamo ammettere nuovi clienti
        if (possiamo_ammettere_clienti(supermercato)) {
            int clienti_ammessi = ammetti_clienti(supermercato);
            if(clienti_ammessi){
            printf(COLOR_BLUE "Ammessi %d nuovi clienti. Clienti attualmente presenti: %d.\n" COLOR_RESET, clienti_ammessi, supermercato->clienti_presenti);
            }
            else if (!clienti_ammessi) {
                printf(COLOR_RED "Non ci sono clienti da ammettere o limite massimo raggiunto.\n" COLOR_RESET);
                break;
            }
        } else {
            printf(COLOR_RED "Non ci sono spazi disponibili per nuovi clienti. Clienti attualmente presenti: %d.\n" COLOR_RESET, supermercato->clienti_presenti);
            break;
        }

        // Sblocca il mutex del supermercato
        pthread_mutex_unlock(&supermercato->mutex_supermercato);

        sleep(10);  // Attende 10 secondi prima di ricontrollare
    }

    exit(0);
    return NULL;
}
```

5.3 Client Handler

Listing 3: Client Handler

```
void *client_handler(void *arg) {
    pthread_arg_t *pthread_arg = (pthread_arg_t *)arg;
    int new_socket_fd = pthread_arg->new_socket_fd;
```

```
4       Supermercato *supermercato = pthread_arg->supermercato;
5       free(arg);
6
7       //detached thread
8       pthread_detach(pthread_self());
9
10      char buffer[BUFFER_SIZE] = {0};
11
12      while (read(new_socket_fd, buffer, BUFFER_SIZE) > 0) {
13          char command[20] = {0};
14          sscanf(buffer, "%s", command);
15
16          Cliente cliente;
17          cliente.id = new_socket_fd; // utilizzo dell'ID
    socket per rappresentare univocamente il cliente
18          switch (command[0]) {
19              case 'E':
20                  if (strncmp(buffer, "ENTRY_REQUEST", 13) ==
    0) {
21                      int time_to_shop = 0, num_items = 0;
22                      if (sscanf(buffer + 14, "%d %d", &
    time_to_shop, &num_items) != 2) {
23                          close(new_socket_fd);
24                          return NULL;
25                      }
26                      cliente.tempo_per_scegliere_oggetti =
    time_to_shop;
27                      cliente.numero_di_oggetti = num_items;
28
29                      pthread_mutex_lock(&supermercato->
    mutex_supermercato);
30                      if (supermercato->clienti_fuori <
    supermercato->max_clienti) {
31                          write(new_socket_fd, "ENTRY_ACCEPTED"
    , strlen("ENTRY_ACCEPTED"));
32                          supermercato->lista_attesa[
    supermercato->clienti_fuori++] = &cliente;
33                      } else {
34                          write(new_socket_fd, "ENTRY_DENIED",
    strlen("ENTRY_DENIED"));
35                      }
36                      pthread_mutex_unlock(&supermercato->
    mutex_supermercato);
37                  }
38              case 'Q':
39              break;
40              case 'P':
41                  if (strncmp(buffer, "PAYMENT_REQUEST", 15) ==
     0) {
```

```
42                      printf(CYAN_COLOR "Il cliente %d ha
    effettuato il pagamento\n" RESET_COLOR, cliente.id);
43                  }
44                  break;
45              case 'N':
46                  if (strncmp(buffer, "NO_ITEMS_EXIT_REQUEST",
    21) == 0) {
47                      printf(RESET_COLOR "Il cliente %d ha
    deciso di uscire senza acquistare nulla\n" RESET_COLOR,
    cliente.id);
48                  }
49                  break;
50              default:
51                  printf("Comando non riconosciuto: %s\n",
    buffer);
52                  break;
53          }
54          memset(buffer, 0, BUFFER_SIZE);
55      }
56
57      close(new_socket_fd);
58      pthread_exit(NULL);
59      return NULL;
60  }
```

5.4 Activity Diagram per la funzione client_handler

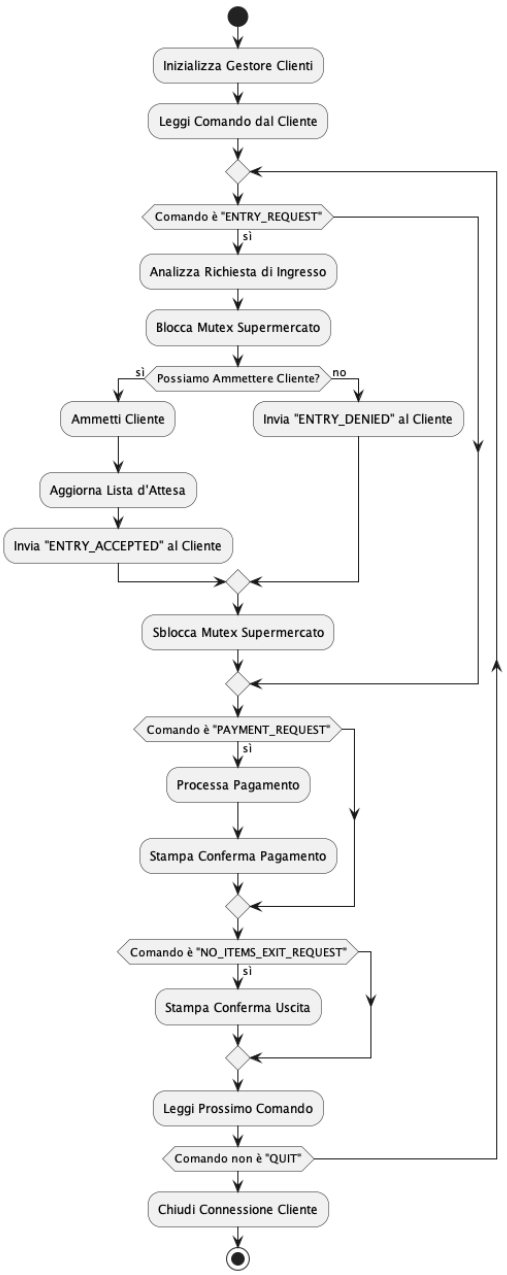

Figure 1: Activity Diagram di client_handler