Università degli Studi di Enna "Kore"
Facoltà di Ingegneria e Architettura
Corso di Studi in Ingegneria Informatica

Sistemi Operativi

RELAZIONE ELABORATO

STUDENTE 1	
Nome	Giovanni
Cognome	Pennisi
Matricola	23038584
Email	giovanni.pennisi@unikorestudent.it

STUDENTE 2	
Nome	Lorenzo Calogero
Cognome	Fretto
Matricola	23038049
Email	lorenzocalogero.fretto@unikorestudent.it

SPECIFICHE ELABORATO	
Titolo	*[dare un titolo al progetto]* Sistema ospedaliero digitalizzato
Descrizione dettagliata	*[descrivere molto dettagliatamente l'intero elaborato]* L'elaborato si pone come obiettivo quello di simulare un sistema computerizzato di gestione di un centro medico ospedaliero (pronto soccorso); pensato per essere molto versatile, dà la possibilità di gestire più medici e più pazienti ed è suddiviso in due parti, il lato client ed il lato server: Nel *lato server*, vengono gestite le operazioni relative all'arrivo fisico dei pazienti al pronto soccorso (il quale viene gestito in modo asincrono, con ogni paziente che arriva dopo un tempo casuale compreso tra 1 e 5 secondi, gestito tramite una sleep del thread). L'addetto agli arrivi li registra assegnando loro un codice di priorità (assegnato in modo randomico, per simulare appunto l'operatore che svolge il pre-triage) tra bianco, verde, azzurro, arancione e rosso. Le priorità sono gestite tramite 5 code, una per grado di priorità, implementate utilizzando il principio delle FIFO (first in, first out). Successivamente i pazienti vengono indirizzati nelle "sale d'attesa", pronti per essere assistiti (le code rappresentano le sale d'attesa). Il *lato client* simula l'attività dei medici presenti nel centro medico. Ognuno di loro è rappresentato da un processo ed è responsabile di una delle cinque priorità. I medici possono fornire assistenza ai pazienti assegnati loro, richiedere aiuto al client del Medico Primario come in caso di problematiche gravi o emergenze (la richiesta d'aiuto viene gestita tramite i segnali tra processi e il Medico Primario è identificato dal file denominato "primario" - ATTENZIONE: è fondamentale che il programma Primario venga eseguito prima degli altri client), dimettere i pazienti

	dopo averli curati (o comunque dopo aver svolto l'attività prevista) e infine possono visualizzare l'elenco di tutti i pazienti dimessi o ancora presenti in struttura. Ricapitolando, il programma mira a simulare il flusso di pazienti attraverso un centro medico, assegnando loro una priorità casualmente, come se fosse in base alla gravità delle condizioni, e consentendo ai medici di interagire con il sistema in modo realistico.
System call utilizzate	*[specificare quali sono le system call scelte e come vengono utilizzate all'interno dell'elaborato]* ❏ processi (*fork*, *exit*): gestione connessione client e server, esecuzione attività specifiche per i pazienti, gestione execl; ❏ *wait* e *waitpid*: per i client, gli accessi e per la terminazione ❏ *exec*: esecuzione di un processo tramite la sostituzione dell'immagine del processo figlio con un nuovo programma, per la gestione del menù e con un'interfaccia grafica personalizzata, passa e gestisce delle variabili. ❏ segnali tra processi: invio da parte dei client dei medici normali una richiesta di assistenza (utilizzando SIGUSR1) al medico Primario. ❏ shared memory: le 5 code per i 5 indici di priorità vengono lette dai client tramite la memoria condivisa ❏ gestione dei thread (*create, self, exit, cancel, join*, attributi): orologio, arrivo pazienti, attività di gestione. ❏ semafori POSIX con nome per thread: più pazienti che arrivano contemporaneamente ❏ mutex per thread: regolare accesso e sincronizzazione (orologio, pazienti, etc) ❏ variabili condizione per thread: raggiungimento capienza massima ospedale (evitare core dump)

Funzioni implementate	*[descrivere singolarmente le varie funzioni implementate all'interno dell'elaborato specificando i parametri di ingresso/uscita e se interagiscono con processi/thread o altre funzioni]*
	
	## Lato Client:
	
	pid_t getPIDByName(const char* processName);
	Restituisce il pid del medico primario.
	
	void inizializzaCoda(CodaPazienti* coda, int capacita, int dimessiCapacita);
	
	int inizializzaConnessione();
	Restituisce il socket del client.
	
	void inviaRichiesta(int client_socket, int client_pid, Paziente* pazienti, int priocode);
	
	void stampaElencoPazientiPerPriorita(CodePriorita* code);
	
	void stampaElencoPazientiSpecifici(CodePriorita* code, int prio);
	
	int getPaziente(CodaPazienti* coda, Paziente* paziente);
	Restituisce 1 se la coda è piena, altrimenti 0 se l'operazione è andata a buon fine.

	void dimettiPaziente(CodaPazienti* coda, Paziente* paziente);
	
	void stampaElencoPazientiDimessi(CodaPazienti* coda);
	
	void gestioneSegnaleUSR1(int sig, siginfo_t* info, void* context);
	
	void inviaSegnalePersonalizzato(int client_pid);
	
	void exec(int codicePrioritaClient, int checkPrimario, int terzoParametro);
	
	void stampaCodiceMenu(int cod, int checkPrimario);
	
	char getch();
	Restituisce il carattere premuto con la tastiera senza la necessità di premere invio.
	
	## Lato Server:
	
	void inizializzaCoda(CodaPazienti* coda, int capacita);
	
	void inizializzaCodePriorita(CodePriorita* code, int capacita);
	
	void inserisciInCoda(CodaPazienti* coda, Paziente* paziente);
	
	void inserisciInCodaPriorita(CodePriorita* code, Paziente* paziente);

	Paziente estraiDaCoda(CodaPazienti* coda);
	Restituisce pazienteEstratto di tipo Paziente, cioè il paziente estratto dalla coda.
	
	Paziente estraiDaCodaPriorita(CodePriorita* code, CodicePriorita priorita);
	Richiama estraiDaCoda passandogli la priorità e quindi scegliendo la coda corrispondente.
	
	void rimuoviPaziente(CodePriorita* code, Paziente elencoPazienti[][5], int* numPazienti, Paziente pazienteRim);
	
	bool confrontaPazienti(const Paziente* p1, const Paziente* p2);
	Restituisce il risultato di strcmp
	
	int generaNumeroCasuale(int minimo, int massimo);
	Restituisce il numero randomico del codice priorità
	
	void inserisciPaziente(CodaPazienti* coda, Paziente elencoPazienti[][5], int* numPazienti, Paziente* paziente);
	
	void inserisciPazientePriorita(CodePriorita* code, Paziente elencoPazienti[][5], int* numPazienti, Paziente* paziente);
	
	Paziente caricaCasuale(FILE* file, CodePriorita* code, Paziente elencoPazienti[][5], int* numPazienti);
	Restituisce il paziente caricato.
	
	void* threadCaricaCasuale(void* args);

	CodicePriorita assegnaPrioritaCasuale();
	Restituisce la priorità scelta casualmente.
	
	void gestisciConnessioneClient(int client_socket, CodePriorita* code, Paziente elencoPazienti[][5], int* numPazienti);
	
	void gestisciServer(CodePriorita* code, Paziente elencoPazienti[][5], int* numPazienti);
	
	int create_shared_memory(size_t size);
	Restituisce shmid (l'id della memoria condivisa)
	
	void* attach_shared_memory(int shmid);
	
	void initialize_semaphores();
	
	void distruggiSemafori(CodePriorita* codePriorita);
	
	void liberaMemoriaCondivisa();
	
	void tempoScorre();
	
	void* orologio(void* args);
	
	void gotoxy(int x, int y);
	
	Le funzioni sviluppate, contestualmente al loro utilizzo nei programmi, sono corredate da commenti esplicativi.

	Viene definita un'enumerazione `CodicePriorita` che rappresenta i vari codici di priorità che i pazienti possono avere. *Viene dichiarata la struttura `Paziente` che prevede le informazioni sul paziente quali nome, cognome, provincia, età e priorità.* *La struttura `CodaPazienti` è per rappresentare una coda condivisa tra i thread e contiene un array di pazienti, informazioni sulla capacità, la testa, la coda e il numero di elementi, e un semaforo per la sincronizzazione dell'accesso (garantire l'accesso mutuamente esclusivo).* *CodePriorita definisce un array di code (5, una per ogni priorità) ordinato gerarchicamente.*