Laboratori 4 de SIOP
Ixent Cornella

1. Capitol

A diferència de l'anterior pràctica, ara el capitol es troba en un servidor remot, no el creem nosaltres. Això significa que necessitarem un socket, per poder comunicar els nostres replicants amb el capitol. El codi per configurar el socket ens es proporcionat per la resistència.

2. TT

a) El TT (programa principal) s'encarrega de preparar els canals, i rediccionar-los si fa falta. Al tractar-se d'un socket, el TT es connecta al servidor remot, de fet ho fa amb dos canals, un per rebre dades (que anirà als replicants) i un de control, per veure si la missió ha tingut èxit. El TT llavors crea M replicants, que en aquest cas, son processos fills. Aquests atacaran al banc del temps, amb funcions que ens proporcionen, mitjançant el canal que hem obert per rebre dades. Abans de res, els replicants redirigeixen el seu canal de lectura al de lectura del socket anteriorment creat. D'aquesta manera, els replicants ho llegeixen i converteixen les dades en temps amb les funcions donades. Finalment, aquests fan exit amb un codi particular, que després es va sumant al dels altres replicants per veure si el procés s'ha executat bé. Si algun dels replicants té un error, llavors s'imprimeix un missatge d'error i s'aborta el programa.

b)

```
TT:
fd = obtenir_canal_dades()
fdc = obtenir_canal_control()
for(M replicants){
        redirgeix_canals(fd, 0)
        inicialitza_contenidor
        while(llegeix_del_socket){
                calcula_temps;
                }
        dummy_exit()
 }
 if(!error) suma_codis = suma_codis + codi //M vegades

else: print("Error amb els replicants!!")
```

c)

```
//CODI
int acum = 0;

    int ctrlfd = ctrlconnect(SERVADDR, CTRLPORT, ENIGMA3, TEAMNAME, SILVERKEY, GOLDKEY); //retorna el file descriptor del canal de control i verifica l'enigma i la clau
    int fd = connecta(SERVADDR, SERVPORT); //retorna el file descriptor del canal de dades

    //SEGUEIX A LA SEGUENT PAGINA
```

```
	//creem M replicants.
	for(int i = 0; i < M; ++i){

		int id = fork();
		if(id == 0){
			//redirigim els canals del replicant.
			dup2(fd, 0);
			close(fd);
			close(ctrlfd);
			// executem per cada replicant el codi per atacar el banc
			execl("./replicant", "./replicant", argv[2], NULL);

		}
	}

	//esperem els replicants...
	int st;
	while(waitpid(-1, &st, 0) > 0){
		if (WIFEXITED(st)) {
			int es = WEXITSTATUS(st);
			//printf("Exit status was %d\n", es);
			acum += es;
		}
		else{
			printf("ERROR: 'N' unitats de temps son massa grans per poder-les
robar-les. Redueix N.\n");
			while(wait(NULL) > 0); //ABORTEM MISSIO
		}
	}

	//Enviem acum al capitol, i rebem la seva resposta en funcio de si acum esta be o
no.
	/*******sending acum to capitol ******************/
	char buf[10];
	int nbytes = sprintf (buf, "%d", acum);
	write(ctrlfd,buf,nbytes);

	/*******reading answer from Capitol **************/
	char buffy[128];
	int len = read(ctrlfd,buffy,128);
	buffy[len] = '\0';
	printf("MISSION: %s\n",buffy);
```

3. Replicants

a) Els replicants ara no ataquen de forma local el capitol, sinó que la conexió es remota. El codi d'aquest pràcticament no canvia, doncs els replicants sempre tindran el canal de lectura estàndard redirigit al canal de lectura de dades, el socket. Llavors, el codi serà pràcticament el mateix que l'anterior pràctica.

b)

```
if(N es correcta){
        inicialitza_dummy(buffer, N)
        mentre(quedin dades per llegir){
                llegeix_en_blocs(N, buffer)
                transforma_temps(buffer)
        }
        dummy_end()
}
else error().
```

c)

```
int N = atoi(argv[1]);
char buffer[N];         //Aqui processarem el "temps".

//Comprovem que sigui correcte el tamany del buffer.
if(N > 0 && N <= MAXTAM){
        int x;
        dummy_init(buffer, N);
        //robem el temps i el processem.
        while((x = read(0, &buffer, N)) > 0)
                dummy_calc(buffer, x);
        dummy_end();
}
else {
        exit(-1);
}
```