iplik yaratılması ve birleştirilmesi işlemleri

```
/*
    Creating and joining threads
*/
#define _GNU_SOURCE
#define _REENTRANT
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>

int MAX=5;

int    my_rand( int, int );
void  *say_it( void * );


int main(int argc, char *argv[])
{
    pthread_t   thread_id[MAX];
    int      status, *p_status = &status;
    int      i;

    setvbuf(stdout, (char *) NULL, _IONBF, 0);

    if ( argc > MAX+1 )
    {                // check arg list
        fprintf(stderr, "%s arg1, arg2, ... arg%d\n", argv[0], MAX);
        return 1;
    }

    printf("Displaying\n");

    for (i = 0; i < argc-1; ++i)
    {   // generate threads
        if( pthread_create(&thread_id[i],NULL,say_it,(void *)argv[i+1]) >
0)
        {
            fprintf(stderr, "pthread_create failure\n");
            return 2;
        }
    }

    for (i=0; i < argc-1; ++i)
    {      // wait for each thread
        if ( pthread_join(thread_id[i], (void **) p_status) > 0)
        {
            fprintf(stderr, "pthread_join failure\n");
            return 3;
        }
        printf("\nThread %d returns %d", thread_id[i], status);

    }

    printf("\nDone\n");
```

```
    return 0;
}

//   Display the word passed a random # of times
void * say_it(void *word)
{
    int i;

    int numb = my_rand(2,6);


    printf("%s\t to be printed %d times.\n", (char *)word, numb);

    sleep(1);

    for (i=0; i < numb; ++i)
    {
        sleep(1);
        printf("%s ", (char *) word);
    }
    sleep(4);
    return (void *) NULL;
}

//   Generate a random # within given range
int my_rand(int start, int range)
{
    struct timeval t;
    gettimeofday(&t, (struct timezone *)NULL);
    return (int)(start+((float)range * rand_r((unsigned *)&t.tv_usec)) /
(RAND_MAX+1.0));
}
```

ipliklerde global değişkenlerin kullanılması durumu

```
/*
         Using global variables in multiple threads
*/
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int myglobal;

void *thread_function(void *arg)
{
    int i,j;
    for ( i=0; i<20; i++ )
    {
        j=myglobal;
        j=j+1;
        printf(".");
        fflush(stdout);
        sleep(1);
        myglobal=j;
    }
    return NULL;
}

int main(void)
{

    pthread_t mythread;
    int i;

    if ( pthread_create( &mythread, NULL, thread_function, NULL) )
    {
        printf("error creating thread.");
        abort();
    }

    for ( i=0; i<20; i++)
    {
        myglobal=myglobal+1;
        printf("o");
        fflush(stdout);
        sleep(1);
    }

    if ( pthread_join ( mythread, NULL ) )
    {
        printf("error joining thread.");
        abort();
    }
    printf("\nmyglobal equals %d\n",myglobal);

    exit(0);

}
```

ipliklerin çalışma sırasını devretmesi (yield işlemi)

```
/* yield_count.c */
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>

typedef void *(*create_func)(void *);

static int child_ran = 0;
static int parent_ran = 0;
static int *child_run, *parent_run;

static void *child_function(int *runs)
{
    register int i, run = *runs;

    child_ran = 1;
    for (i = 0; i < run; i++)
    {
        for (child_run[i] = 0; !parent_ran; child_run[i]++)
        {
            sched_yield ();
        }
        parent_ran = 0;
        child_ran = 1;
    }

    return NULL;
}

int main(int argc, char *argv[])
{
    int i, runs;
    pthread_t child_thread;


    if (argc != 2)
    {
        fputs("Usage: yield runs, where runs is the number of times to try
yielding\n", stderr);
        return EXIT_FAILURE;
    }

    i = 1;

    runs = atoi (argv[i]);
    child_run = (int *) malloc (runs * sizeof (int));
    parent_run = (int *) malloc (runs * sizeof (int));

    pthread_create(&child_thread, NULL, (create_func) child_function,
&runs);
    parent_ran = i;
    for (i = 0; i < runs; i++)
    {
```

```
        for (parent_run[i] = 0; !child_ran; parent_run[i])
        {
            sched_yield ();
        }
        child_ran = 0;
        parent_ran = 1;
    }

    pthread_join(child_thread, NULL);

    puts("Parent statistics:");
    for (i = 0; i < runs; i++)
    {
        printf ("  Run %d: %d yields before the child ran\n", i,
parent_run[i]);
    }

    puts("Child statistics:");
    for (i = 0; i < runs; i++)
    {
        printf ("  Run %d: %d yields before the parent ran\n", i,
child_run[i]);
    }

    free (child_run);
    free (parent_run);
    return EXIT_SUCCESS;
}
```

ipliklerin çalışma sırasında iptal edilmesi

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define LOOPS 1024
#define WASTE_TIME (1 << 18)

typedef void *(*create_func)(void *);
typedef void (*cleanup_func)(void *);

void cleanup_function()
{
    puts("Ladies and gentlemen, Elvis has left the building.");
}

void *child_function()
{
    int i;

    pthread_cleanup_push((cleanup_func)cleanup_function, NULL);
    for (i = 0; i < LOOPS; i++)
    {
        printf("And a %d,\n", i);
        pthread_testcancel();
    }
    pthread_cleanup_pop(0);
    puts("Hit it!");
    return NULL;
}

int main(int argc, char *argv[])
{
    int sum, i;
    pthread_t child_thread;

    sum = 0;

    pthread_create(&child_thread, NULL, (create_func)child_function, NULL);
    puts("I think I'll waste a little time.  Do you mind?");
    for (i = 0; i < WASTE_TIME; i++)
    {
        sum += i;
    }


    pthread_cancel(child_thread);
    return EXIT_SUCCESS;
}
```