```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>

#define MAX_STOCK 100
#define SIMULATION_TIME 30
#define CONSUMER_INTERVAL 2
#define PRODUCER_INTERVAL 5

typedef struct
{
    int id;
    int amount;
} ThreadData;

static int stock = 0;
static pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t cond_empty = PTHREAD_COND_INITIALIZER;
static pthread_cond_t cond_full = PTHREAD_COND_INITIALIZER;
static int simulation_running = 1;
static int current_time = 0;
static int consumer_id = 1;
static int producer_id = 1;
static int waiting_consumers = 0;
static int waiting_producers = 0;

void table(const char* type, int current_time, int id, int requested, int
processed, const char* waitlist, int remaining)
{
    printf("\n%s Activity\n", type);
    printf("Time\tID\tAmount\tProcessed\tWaitlist?\tRemaining\tStock\n");
    printf("%d\t%d\t%d\t%d\t\t%s\t\t%d\t\t%d\n", current_time, id, requested,
processed, waitlist, remaining, stock);
}

void* consumer(void *arg)
{
    ThreadData *data = (ThreadData*)arg;
    int id = data->id;
    int amount = data->amount;
    free(data);

    pthread_mutex_lock(&mtx);
    int processed = 0;
    int remaining = amount;
    int print_table = 1;

    while (remaining>0 && simulation_running)
    {

        if (stock > 0)
        {
            int purchase = (stock < remaining) ? stock : remaining;
            stock -= purchase;
            processed += purchase;
            remaining -= purchase;
            table("Consumer", current_time, id, amount, processed, remaining > 0 ?
"yes" : "no", remaining);
            amount = remaining;
            processed = 0;
            print_table = 0;
            if (stock < MAX_STOCK)
            {
                pthread_cond_broadcast(&cond_full);
            }
        }
        else
        {
            waiting_consumers++;
            if (print_table)
            {
                table("Consumer", current_time, id, amount, processed, "yes",
remaining);
            }
            pthread_cond_wait(&cond_empty, &mtx);
            waiting_consumers--;
            print_table = 1;
        }
    }

    pthread_mutex_unlock(&mtx);
    return NULL;
}

void* producer(void *arg)
{
    ThreadData *data = (ThreadData*)arg;
    int id = data->id;
    int amount = data->amount;
    free(data);

    pthread_mutex_lock(&mtx);
    int processed = 0;
    int remaining = amount;
    int print_table = 1;

    while (remaining > 0 && simulation_running)
    {
        if (stock < MAX_STOCK)
        {
            int space_available = MAX_STOCK - stock;
            int delivery = (space_available < remaining) ? space_available :
remaining;
            stock += delivery;
            processed += delivery;
            remaining -= delivery;

            table("Producer", current_time, id, amount, processed, remaining > 0 ?
"yes" : "no", remaining);
            amount = remaining;
            processed = 0;
            print_table = 0;

            if (stock > 0 && waiting_consumers > 0)
            {
                pthread_cond_broadcast(&cond_empty);
            }

            if (stock < MAX_STOCK && waiting_producers > 0)
            {
                pthread_cond_broadcast(&cond_full);
            }
        }
        else
        {
            waiting_producers++;
            if(print_table)
            {
                table("Producer", current_time, id, amount, processed,  "yes", 
remaining);
            }
            pthread_cond_wait(&cond_full, &mtx);
            waiting_producers--;
            print_table = 1;
        }
    }

    pthread_mutex_unlock(&mtx);
    return NULL;
}

int main()
{
    srand(time(NULL));
    pthread_t threads[100];
    int thread_count = 0;

    printf("\n=== Producer-Consumer Simulation ===\n");
    printf("Simulation Time: %d seconds\n", SIMULATION_TIME);
    printf("Consumer Interval: Every %d seconds\n", CONSUMER_INTERVAL);
    printf("Producer Interval: Every %d seconds\n", PRODUCER_INTERVAL);
    printf("Max Stock Capacity: %d units\n\n", MAX_STOCK);

    for (current_time = 0; current_time <= SIMULATION_TIME; current_time++)
    {
        if(current_time > 0 )
        {
            if (current_time % CONSUMER_INTERVAL == 0)
            {
                ThreadData *data = malloc(sizeof(ThreadData));
                data->id = consumer_id++;
                data->amount = rand() % 5 + 1;
                pthread_create(&threads[thread_count++], NULL, consumer, data);
            }

            if (current_time % PRODUCER_INTERVAL == 0)
            {
                ThreadData *data = malloc(sizeof(ThreadData));
                data->id = producer_id++;
                data->amount = rand() % 20 + 1;
                pthread_create(&threads[thread_count++], NULL, producer, data);
            }

            sleep(1);
        }
    }

    pthread_mutex_lock(&mtx);
    simulation_running = 0;
    pthread_cond_broadcast(&cond_empty);
    pthread_cond_broadcast(&cond_full);
    pthread_mutex_unlock(&mtx);

    for (int i = 0; i < thread_count; i++)
    {
        pthread_join(threads[i], NULL);
    }

    printf("\n\n=== Results ===\n");
    printf("Stock: %d\n", stock);
    printf("Total Consumers Created: %d\n", consumer_id-1);
    printf("Total Producers Created: %d\n", producer_id-1);
    printf("Waiting Consumers: %d\n", waiting_consumers);
    printf("Waiting Producers: %d\n\n", waiting_producers);

    return 0;
}
```