|  |  |
| --- | --- |
| CS 241 | Lecture#12 Creating Threads |

#1 Sketch in memory all of the places there is a variable "c"

void recurse(int param) {

int c= 10;

c ++;

if(param >1) recurse(param-1);

}

void\* start(void\* ptr) {

recurse(3);

return NULL;

}

#define NTHREADS (3)

int main() {

pthread\_t tids[NTHREADS];

for(int i=0;i< NTHREADS;i++) {

? \_\_\_\_

}

pthread\_exit(NULL); // No more after here!

return 42;

}

#2 Independent Threads

#define N (10)

pthread\_t tid\_runners[N];

pthread\_t tid\_display;

int width;

int height;

int main() {

getTerminalWidthHeight(&width, &height);

void\* image = malloc(height \* width);

**memset**(image, '.' , height \* width);

pthread\_create( **& tid\_display**,NULL,display, image);

for(int i=0;i<N;i++)

pthread\_create(**tid\_runners+i**,NULL,run, image);

**pthread\_exit**(NULL);

return 42; // so we will never know the answer

}

void\* display(void\* ptr) {

while(1) {

for(int y=0; y < height;y++) {

write( 1, ptr + y \* width, width);

write( 1, "\n", 1);

}

// Move back up by height lines then sleep for 10ms

for(int y=0; y < height; y++) write(1,"\033[1A",5);

usleep(10000); // 10 millisecond delay

}

return NULL;

}

void\* run(void\* ptr) {

char\* image = ptr;

while(1) {

int x = rand() % width; // random column

for(int y= 0 ; y < height && ... ; y++ ) {

image[ x + y\*width ] ^= 0x6E; // flip bits

usleep(x \* 5000);

}

}

return NULL;

}

#3 Can threads access heap memory? Can one thread malloc and another free?

#4 What is your question about threads and processes?

#5 Case study: Embarrasingly ||, no-IO, Mandelbrot Set

uint32\_t\* myPixels = calloc(width \* height,sizeof(uint32\_t));

for(int y=0; y < height; y++) {

for(int x=0; x < width; x++) {

myPixels[x + y \* width] = mandelbrot(x,y);

}

// update the window every 16 rows

if((y & 0xf) == 0xf) update\_gui(); // direct coupling

}

uint32\_t mandelbrot(int x, int y) {

double const complex c = *realVal(x)* + I \* *imgVal(y)*;

double complex z = 0;

int iterations = 0;

for(; iterations < max\_iterations && cabs(z)< 2;iterations++)

z = z \* z + c;

// Convert the iteration count into the R G B bytes

return (cabs(z) < 2) ? 0xffffff : iterations \* 0x81021;

}

#6 Attempt 1 – pthread all the rows! Hack the void pointer

for(int y = 0; y < height; y++) {

void\* hack = **(void\*) y**;

int r = pthread\_create( & tids[y] , NULL, calc1, hack);

if(r) quit("pthread\_create failed");

}

void\* calc1(void\*hax) {

int y = **(int) hax**; //We are NOT derefencing hax

for(int x=0; x < width; x++)

myPixels[x + y \* width] = mandelbrot(x,y);

}

#7 Attempt2 – Use arg as a real pointer

for(int y = 0; y < height; y++) {

printf("Creating thread....%d\n",y);// Don't delete

int r = pthread\_create( & tids[y], NULL, calc2, **&y**);

if(r) quit("pthread\_create failed");

SDL\_Delay(1); // If it crashes increase this value

}

void\* calc2(void\* better) {

int\* intptr = (int\*) better;

int y = \*intptr;

for(int x=0; x < width; x++) {

myPixels[x + y \* width] = mandelbrot(x,y);

#8 Create task structs & limit max number of threads

typedef struct \_task\_t {

int start\_x;

int start\_y;

int end\_x;

int end\_y;

} task\_t;

num\_tasks =((height+63)/64) \* ((width+63)/64);

task\_t\* tasks = calloc(num\_tasks , sizeof(task\_t));

for(int y = 0; y < height; y+= 64){

for(int x = 0; x < width; x+= 64) {

tasks[i].start\_x = x;

tasks[i].start\_y = y;

tasks[i].end\_x = min(x+size,width);

tasks[i].end\_y = min(y+size,height);

i++;

}

}

void run\_all\_tiles\_and\_wait() {

for(int i= 0; i < num\_tasks; i++) {

pthread\_create( & thread\_ids[thread\_count++]

, NULL, calc3, tasks+i);

if(thread\_count == max\_threads||i+1 == num\_tasks) {

for(int i=0;i< thread\_count; i++) {

pthread\_join(thread\_ids[i], NULL);

}

thread\_count = 0; update\_gui();

}

}

} // Most efficient?

void\* calc3(void\* arg) {

task\_t\* task = (task\_t\*) arg;

for(int x = task->start\_x; x < task->end\_x; x++)

for(int y = task->start\_y; y < task->end\_y; y++)

myPixels[x + y \* width] = mandelbrot(x,y);

}