Advanced Operating Systems (2018)

Homework 4 Report & Code

Time taken by unique_ptr implementation:

Number of nodes: 1000000

Time taken during insert: 126ms

Time taken during delete: 1853ms

Time taken during cleanup: 28ms

Time taken by shared_ptr implementation:

Number of nodes: 1000000

Time taken during insert: 139ms

Time taken during delete: 3154ms

Time taken during cleanup: 71ms

Time taken by raw pointer implementation:

Time taken during insert: 49ms

Time taken during delete: 1692ms

Time taken during cleanup: 0ms

The shared_ptr takes the most amount of time while raw pointer takes the least amount of time. Shared pointer takes the most time because there can by multiple references to a variable and hence ownership is not constant.

Segmentation Fault for unique_ptr and shared_ptr

When you run unique_ptr and shared_ptr for 10^6 nodes, the program is likely to crash. The segmentation fault occurs during the cleanup process. The cleanup process in case of smart pointers is done by automatic garbage collector; the garbage collector calls the destructor function to clear the memory. In case of smart pointer implementation of linked list the destructor is implemented in a recursive fashion, i.e., to delete one node, it's next node is deleted and so on. This leads to usage of stack space, the implementation works fine till the number of nodes are within the stack memory, once the number of nodes exceeds the size of the stack, your program crashes and you get segmentation fault. To mitigate this problem, we

can override the destructor; we change its implementation from recursive to iterative fashion. Doing this can solve the problem of segmentation fault.

Code:

unique_shared.cpp

```
#include <iostream>
#include <memory>
#include <unistd.h>


// compile: g++ -std=c++14 -O3 unique.cpp -o unique

static int numinserts = 1000000;

struct node {
	int val;
	std::shared_ptr<struct node> next;
	~node()
	{
		std::shared_ptr<node> temp = std::move(next);
		while(temp!=NULL)
		{
			temp = std::move(temp->next);
		}

	}
};

static unsigned long long rdtsc()
{
	unsigned hi, lo;

	asm volatile("rdtsc" : "=a"(lo), "=d"(hi));
	return (((unsigned long long)hi) << 32) | lo;
}

int delete_node(std::shared_ptr<struct node>& root, int idx)
{
	if (root == NULL || idx < 0)
		return 0;

	if (idx == 0) {
		root = std::move(root->next);
		return 1;
	}

	std::shared_ptr<struct node> *rootp = &root;
```

```
	while (*rootp) {
		if (idx == 1) {
			std::shared_ptr<struct node> todel = std::move((*rootp)-
>next);
			if (todel)
				{
				(*rootp)->next = std::move(todel->next);
		//todel = NULL;
	}
			return 1;
		}
		rootp = &(*rootp)->next;
		idx--;
	}
	return 0;
}

int iterate_list(std::shared_ptr<struct node>* root)
{
	if (root == NULL)
		return 0;

	std::shared_ptr<struct node> rootp = *root;
	int ret = 0;

	while (rootp) {
		ret++;
		rootp = (rootp)->next;
	}
	return ret;
}

int insert_nodes(std::shared_ptr<struct node>& root)
{
	std::shared_ptr<struct node> elem;
	elem = std::shared_ptr<struct node>(new(struct node));
	if (elem == NULL)
		return 0;
	elem->val = 0;
	elem->next = (root);
	root = (elem);
	return 1;
}

int main(int argc, const char *argv[])
{
	std::shared_ptr<struct node> head = NULL;
	int numelems;
	unsigned long long time;
```

```
	if (argc == 2) {
		numinserts = atoi(argv[1]);
	}
	printf("numinserts : %d\n", numinserts);
	if (numinserts < 0)
		return 0;

	time = rdtsc();
	for (int i = 0; i < numinserts; i++) {
		if (!insert_nodes(head)) {
			std::cout << "failed to insert node : " << i << std::endl;
			return 0;
		}
	}
	printf("time taken during insert: %lld ms\n", (rdtsc() - time)/2530000);

	numelems = iterate_list(&head);
	if (numelems != numinserts) {
		printf("test failed! %d %d\n", numelems, numinserts);
	}

	time = rdtsc();
	for (int i = 0; i < numinserts/20; i++) {
		int idx = rand() % ((numinserts/20) - i);
		if (!delete_node(head, idx)) {
			std::cout << "failed to delete node : " << idx << std::endl;
			return 0;
		}
	}
	printf("time taken during delete: %lld ms\n", (rdtsc() - time)/2530000);

	numelems = iterate_list(&head);
	if (numelems != numinserts - (numinserts/20)) {
		printf("test failed! %d %d\n", numelems, numinserts);
	}

	time = rdtsc();
	head = NULL;
	printf("time taken during cleanup: %lld ms\n", (rdtsc() - time)/2530000);


	return 0;
}
```

unique_raw.cpp

```
#include <iostream>
#include <memory>
```

```
#include <unistd.h>
#include <cstdlib>


// compile: g++ -std=c++14 -O3 unique.cpp -o unique

static int numinserts = 1000000;

struct node {
	int val;
	struct node *next;
};

static unsigned long long rdtsc()
{
	unsigned hi, lo;

	asm volatile("rdtsc" : "=a"(lo), "=d"(hi));
	return (((unsigned long long)hi) << 32) | lo;
}

int delete_node(struct node **root, int idx)
{
	if (*root == NULL || idx < 0)
		return 0;

	if (idx == 0) {
		(*root) = (*root)->next;
		return 1;
	}

	struct node *rootp = *root;

	while (rootp!=NULL) {
		if (idx == 1) {
			struct node *todel = rootp->next;
			if (todel!=NULL)
				(rootp)->next = todel->next;
			return 1;
		}
		rootp = rootp->next;
		idx--;
	}
	return 0;
}

int iterate_list(struct node **root)
{
	if (root == NULL)
		return 0;
```

```
	struct node *rootp = *root;
	int ret = 0;

	while (rootp!=NULL) {
		ret++;
		//std::cout << (rootp->val) <<"\n";
		rootp = rootp->next;
	}
	return ret;
}

int insert_nodes(struct node ** root)
{
	//std::unique_ptr<struct node> elem;
	struct node *elem = NULL;
	elem =
	elem = (struct node*)malloc(sizeof(struct node*));
	if (elem == NULL)
		return 0;
	elem->val = 0;
	elem->next = *root;
	*root = elem;
	return 1;
}

int main(int argc, const char *argv[])
{
	struct node *head = NULL;
	int numelems;
	unsigned long long time;

	if (argc == 2) {
		numinserts = atoi(argv[1]);
	}
	printf("numinserts : %d\n", numinserts);
	if (numinserts < 0)
		return 0;

	time = rdtsc();
	for (int i = 0; i < numinserts; i++) {
		if (!insert_nodes(&head)) {
			std::cout << "failed to insert node : " << i << std::endl;
			return 0;
		}
	}
	printf("time taken during insert: %lld ms\n", (rdtsc() - time)/2530000);

	numelems = iterate_list(&head);
```

```
	if (numelems != numinserts) {
		printf("test failed! %d %d\n", numelems, numinserts);
	}

	time = rdtsc();
	for (int i = 0; i < numinserts/20; i++) {
		int idx = rand() % ((numinserts/20) - i);
		if (!delete_node(&head, idx)) {
			std::cout << "failed to delete node : " << idx << std::endl;
			return 0;
		}
	}
	printf("time taken during delete: %lld ms\n", (rdtsc() - time)/2530000);

	numelems = iterate_list(&head);
	if (numelems != numinserts - (numinserts/20)) {
		printf("test failed! %d %d\n", numelems, numinserts);
	}

	time = rdtsc();
	head = NULL;
	printf("time taken during cleanup: %lld ms\n", (rdtsc() - time)/2530000);


	return 0;
}
```