```
 1: // $Id: prodconsbuf3.cpp,v 1.29 2014-06-04 12:06:43-07 - - $
 2:
 3: // Producer/consumer problem using mutex and condition_variable.
 4:
 5: #include <array>
 6: #include <cassert>
 7: #include <chrono>
 8: #include <condition_variable>
 9: #include <iomanip>
10: #include <iostream>
11: #include <mutex>
12: #include <sstream>
13: #include <string>
14: #include <thread>
15: #include <vector>
16: using namespace std;
17:
18: //
19: // Timer.
20: //
21: class elapsed_time {
22:    private:
23:       using clock = chrono::high_resolution_clock;
24:       clock::time_point start {clock::now()};
25:    public:
26:       double elapsed_nanoseconds() {
27:          clock::time_point now = clock::now();
28:          return chrono::duration_cast<chrono::nanoseconds> (now - start)
29:                 .count() / 1e9;
30:       }
31: } timer;
32:
```

```
33:
34: //
35: // Printer for synchronized output accepts a variable number
36: // of arguments.
37: //
38:
39: class synch_printer {
40:    private:
41:       mutex out_mutex;
42:       ostream& out;
43:       void print_();
44:       template <typename Head, typename... Tail>
45:       void print_ (const Head& head, Tail... tail);
46:    public:
47:       synch_printer (ostream& out): out(out){}
48:       template <typename... Type>
49:       void print (Type... params);
50: };
51:
52: void synch_printer::print_() {
53: }
54:
55: template <typename Head, typename... Tail>
56: void synch_printer::print_ (const Head& head, Tail... tail) {
57:    out << head;
58:    print_ (tail...);
59: }
60:
61: template <typename... Type>
62: void synch_printer::print (Type... params) {
63:    unique_lock<mutex> lock {out_mutex};
64:    out << setw(9) << setprecision(6) << fixed
65:        << timer.elapsed_nanoseconds() << " ";
66:    print_ (params...);
67:    out << endl << flush;
68: }
69:
70: //
71: // Trace block/function entry/exit.
72:
73: class start_trace {
74:    private:
75:       const string name;
76:       synch_printer& printer;
77:    public:
78:       start_trace (string name, synch_printer& printer):
79:                    name(name), printer(printer) {
80:          printer.print (name, " STARTING");
81:       }
82:       ~start_trace() { printer.print (name, " FINISHED"); }
83: };
84:
```

```
 85:
 86: //
 87: // class bounded_buffer
 88: // NOT synchronized.
 89: // Just your ordinary Data Structures queue.
 90: //
 91:
 92: template <typename Type, size_t size>
 93: class bounded_buffer {
 94:    public:
 95:       using value_type = Type;
 96:    private:
 97:       static constexpr ssize_t EMPTY {-1};
 98:       ssize_t head {EMPTY};
 99:       ssize_t tail {EMPTY};
100:       array<Type,size> items;
101:    public:
102:       bool empty() const { return head == EMPTY; }
103:       bool full() { return (tail + 1) % size == head; }
104:       const value_type& front() const;
105:       void pop_front();
106:       void push (const value_type& val);
107: };
108:
109: class bounded_buffer_error: public runtime_error {
110:    public:
111:       explicit bounded_buffer_error (const string& what);
112: };
113:
114: bounded_buffer_error::bounded_buffer_error (const string& what):
115:             runtime_error (what) {
116: }
117:
118: template <typename Type, size_t size>
119: const Type& bounded_buffer<Type,size>::front() const {
120:    if (empty()) throw bounded_buffer_error ("front (empty)");
121:    return items[head];
122: }
123:
124: template <typename Type, size_t size>
125: void bounded_buffer<Type,size>::pop_front() {
126:    if (empty()) throw bounded_buffer_error ("pop_front (empty)");
127:    if (head == tail) head = tail = EMPTY;
128:                 else head = (head + 1) % size;
129: }
130:
131: template <typename Type, size_t size>
132: void bounded_buffer<Type,size>::push (const value_type& val) {
133:    if (full()) throw bounded_buffer_error ("push (full)");
134:    if (empty()) head = tail = 0;
135:           else tail = (tail + 1) % size;
136:    items[tail] = val;
137: }
138:
```

```
139:
140: //
141: // class synchronized_buffer
142: // prevents concurrent access and uses the bounded_buffer.
143: //
144:
145: template <typename Type, size_t size>
146: class synchronized_buffer {
147:    public:
148:       using value_type = Type;
149:    private:
150:       bounded_buffer<Type,size> buffer;
151:       mutex lock;
152:       condition_variable put_wait;
153:       condition_variable get_wait;
154:       bool producers_gone {false};
155:    public:
156:       struct end_data: public exception{};
157:       void put (const value_type& val);
158:       value_type get();
159:       void end_data_notify_all();
160: };
161:
162: template <typename Type, size_t size>
163: void synchronized_buffer<Type,size>::put (const value_type& val) {
164:    unique_lock<mutex> ulock (lock);
165:    while (buffer.full()) put_wait.wait (ulock);
166:    buffer.push (val);
167:    get_wait.notify_one();
168: }
169:
170: template <typename Type, size_t size>
171: Type synchronized_buffer<Type,size>::get() {
172:    unique_lock<mutex> ulock (lock);
173:    while (buffer.empty()) {
174:       if (producers_gone) throw end_data();
175:                      else get_wait.wait (ulock);
176:    }
177:    value_type result = buffer.front();
178:    buffer.pop_front();
179:    put_wait.notify_one();
180:    return result;
181: }
182:
183: template <typename Type, size_t size>
184: void synchronized_buffer<Type,size>::end_data_notify_all() {
185:    unique_lock<mutex> ulock (lock);
186:    producers_gone = true;
187:    get_wait.notify_all();
188: }
189:
```

```
190:
191: //
192: // Producer and consumer threads.
193: //
194:
195: using buf_data = pair<string,int>;
196: using synch_buffer = synchronized_buffer<buf_data,5>;
197: string to_string (const buf_data& data) {
198:    return " [\"" + data.first + "\"," + to_string (data.second) + "]";
199: }
200:
201: void producer (int id, synch_buffer& buffer, synch_printer &mcout,
202:                const vector<string>& words) {
203:    start_trace trace ("producer " + to_string (id), mcout);
204:    for (const auto& word: words) {
205:       this_thread::sleep_for (chrono::milliseconds (id * 300));
206:       buf_data data {word, id};
207:       buffer.put (data);
208:       mcout.print ("producer ", id, to_string (data));
209:    }
210: }
211:
212: void consumer (int id, synch_buffer& buffer, synch_printer &mcout) {
213:    start_trace trace ("consumer " + to_string (id), mcout);
214:    try {
215:       for(;;) {
216:          this_thread::sleep_for (chrono::milliseconds (id * 600));
217:          auto data = buffer.get();
218:          mcout.print ("consumer ", id, to_string (data));
219:       }
220:    }catch (synch_buffer::end_data&) {
221:       mcout.print ("consumer ", id, " caught end_data");
222:    }
223: }
224:
```

```
225:
226: //
227: // Main.
228: //
229:
230: template <typename number, class Function>
231: void for_each (number start, number end, Function fn) {
232:    for (; start != end; ++start) fn (start);
233: }
234:
235: int main() {
236:    synch_printer mcout (cout);
237:    start_trace trace ("main", mcout);
238:    synch_buffer buffer;
239:    vector<thread> producers;
240:    vector<thread> consumers;
241:    vector<string> words {"Hello", "World", "foo", "bar", "baz", "qux"};
242:    for_each (1, 4, [&] (int id) {
243:       producers.push_back (thread (producer, id, ref (buffer),
244:                                    ref (mcout), ref (words)));
245:    });
246:    for_each (4, 7, [&] (int id) {
247:       consumers.push_back (thread (consumer, id, ref (buffer),
248:                                    ref (mcout)));
249:    });
250:    for (auto& t: producers) t.join();
251:    buffer.end_data_notify_all();
252:    for (auto& t: consumers) t.join();
253:    return 0;
254: }
255:
256: //TEST// prodconsbuf3 >prodconsbuf3.out 2>&1
257: //TEST// mkpspdf prodconsbuf3.ps prodconsbuf3.cpp* prodconsbuf3.out
258:
```

```
    1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting prodconsbuf3.cpp
    2: prodconsbuf3.cpp:
    3:       $Id: prodconsbuf3.cpp,v 1.29 2014-06-04 12:06:43-07 - - $
    4: g++ -g -O0 -Wall -Wextra -std=gnu++11 prodconsbuf3.cpp -o prodconsbuf3 -
lglut -lGLU -lGL -lX11 -lm -lrt
    5: rm -f prodconsbuf3.o
    6: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished prodconsbuf3.cpp
```

```
 1:  0.000016 main STARTING
 2:  0.001176 producer 2 STARTING
 3:  0.001221 producer 1 STARTING
 4:  0.001319 producer 3 STARTING
 5:  0.001855 consumer 5 STARTING
 6:  0.001889 consumer 6 STARTING
 7:  0.002018 consumer 4 STARTING
 8:  0.301407 producer 1 ["Hello",1]
 9:  0.601295 producer 2 ["Hello",2]
10:  0.601520 producer 1 ["World",1]
11:  0.901453 producer 3 ["Hello",3]
12:  0.901615 producer 1 ["foo",1]
13:  2.402218 consumer 4 ["Hello",1]
14:  2.402272 producer 2 ["World",2]
15:  3.001942 producer 1 ["bar",1]
16:  3.001998 consumer 5 ["Hello",2]
17:  3.602056 consumer 6 ["World",1]
18:  3.602104 producer 3 ["World",3]
19:  4.802424 consumer 4 ["Hello",3]
20:  4.802531 producer 2 ["foo",2]
21:  6.002091 consumer 5 ["foo",1]
22:  6.002174 producer 1 ["baz",1]
23:  7.202241 consumer 6 ["World",2]
24:  7.202284 producer 3 ["foo",3]
25:  7.202669 consumer 4 ["bar",1]
26:  7.202744 producer 2 ["bar",2]
27:  9.002240 consumer 5 ["World",3]
28:  9.002307 producer 1 ["qux",1]
29:  9.002321 producer 1 FINISHED
30:  9.602874 consumer 4 ["foo",2]
31:  9.602921 producer 2 ["baz",2]
32: 10.802433 consumer 6 ["baz",1]
33: 10.802481 producer 3 ["bar",3]
34: 12.002367 consumer 5 ["foo",3]
35: 12.002420 producer 2 ["qux",2]
36: 12.002447 producer 2 FINISHED
37: 12.003057 consumer 4 ["bar",2]
38: 12.003092 producer 3 ["baz",3]
39: 14.402617 consumer 6 ["qux",1]
40: 14.402661 producer 3 ["qux",3]
41: 14.402673 producer 3 FINISHED
42: 14.403245 consumer 4 ["baz",2]
43: 15.002490 consumer 5 ["bar",3]
44: 16.803465 consumer 4 ["qux",2]
45: 18.002623 consumer 5 ["baz",3]
46: 18.002780 consumer 6 ["qux",3]
47: 19.203740 consumer 4 caught end_data
48: 19.203788 consumer 4 FINISHED
49: 21.002810 consumer 5 caught end_data
50: 21.002869 consumer 5 FINISHED
51: 21.602961 consumer 6 caught end_data
52: 21.602997 consumer 6 FINISHED
53: 21.603049 main FINISHED
```