Sample and Exam Projects for COMP S363F

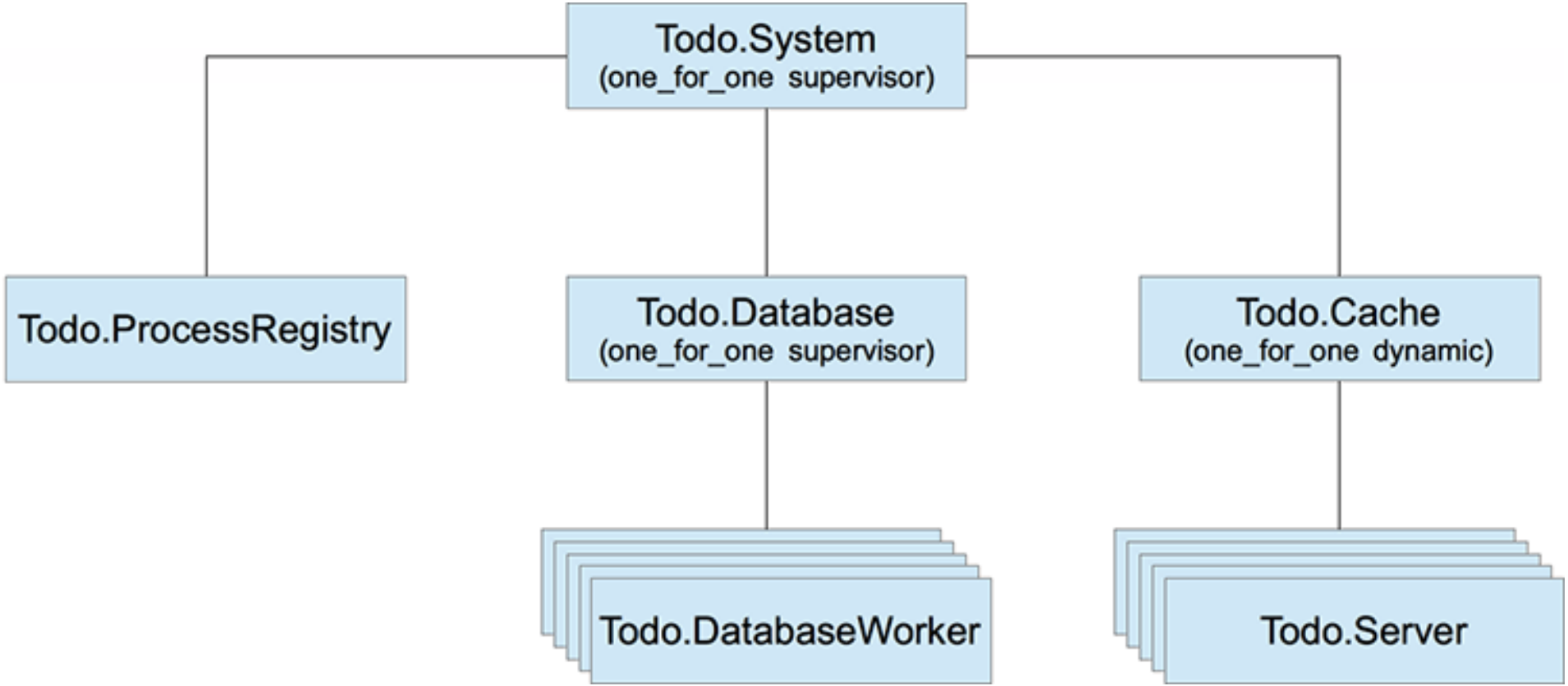
Oliver Au

The Open University of Hong Kong

Outline

1. Supervisor processes in sample project
2. Child processes in sample project
3. Automated unit testing in sample project
4. Exam project’s marking criteria and submission requirements

Sample Project



New process only created when needed

dynamic_workers — beam.smp -- -root /usr/local/Cellar/erlang/22.0....

```
(base) Olivers-MacBook-Pro:dynamic_workers oliverau$ iex -S mix
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:12:12] [ds:12:12:1
0] [async-threads:1] [hipe] [dtrace]

Compiling 7 files (.ex)
warning: Map.size/1 is deprecated. Use Kernel.map_size/1 instead
  lib/todo/list.ex:13

Generated todo app
Interactive Elixir (1.9.0) - press Ctrl+C to exit (type h() ENTER for
 help)
iex(1)> Todo.System.start_link()
Starting database worker 1
Starting database worker 2
Starting database worker 3
Starting to-do cache
{:ok, #PID<0.185.0>}
iex(2)> bobs_list = Todo.Cache.server_process("Bob's list")
Starting to-do server for Bob's list
#PID<0.194.0>
iex(3)> Todo.Cache.server_process("Bob's list")
#PID<0.194.0>
iex(4)> alices_list = Todo.Cache.server_process("Alice's list")
Starting to-do server for Alice's list
#PID<0.197.0>
```

Processes restarted selectively

dynamic_workers — beam.smp -- -root /usr/local/Cellar/erlang...

```
iex(5)> Process.exit(bobs_list, :kill)
true
iex(6)> Todo.Cache.server_process("Bob's list")
Starting to-do server for Bob's list
#PID<0.201.0>
iex(7)> bobs_list = Todo.Cache.server_process("Bob's list")
#PID<0.201.0>
iex(8)> alices_list = Todo.Cache.server_process("Alice's list")
#PID<0.197.0>
iex(9)>
```

1. We mimic the crashing of Bob’s process.
2. If we ask for Bob’s list again, a new process was created so a new ID was returned.
3. If we ask for Alice’s list, no new process was created so the previous ID was returned.
4. Processes handling different todo lists are independent.

dynamic_workers/lib/todo/system.ex

```
defmodule Todo.System do
  def start_link do
    Supervisor.start_link(
      [
        Todo.ProcessRegistry,
        Todo.Database,
        Todo.Cache
      ],
      strategy: :one_for_one
      # Other strategies are :one_for_all and :rest_for_one
    )
  end
end
```

Registry

- A local, decentralized and key-value process storage
- Register and access named processes using `{:via, Registry, {registry, key}}` tuple; see their usage in `DatabaseWorker` module
- Another approach (not shown in our sample code) is to use the 2 functions: `lookup` and `get`
- Process ids are not good after processes are restarted so we use named processes

dynamic_workers/lib/todo/process_registry.ex

```
defmodule Todo.ProcessRegistry do
  def start_link do
    Registry.start_link(keys: :unique, name: __MODULE__)
  end

  def via_tuple(key) do
    {:via, Registry, {__MODULE__, key}}
  end

  def child_spec(_) do
    Supervisor.child_spec(
      Registry,
      id: __MODULE__,
      start: {__MODULE__, :start_link, []}
    )
  end
end
```

dynamic_workers/lib/todo/database.ex (1 of 2)

```
defmodule Todo.Database do
  @pool_size 3
  @db_folder "./persist"

  def start_link do
    File.mkdir_p!(@db_folder)

    children = Enum.map(1..@pool_size, &worker_spec/1)
    Supervisor.start_link(children, strategy: :one_for_one)
  end

  defp worker_spec(worker_id) do
    default_worker_spec = {Todo.DatabaseWorker, {@db_folder, worker_id}}
    Supervisor.child_spec(default_worker_spec, id: worker_id)
  end

  def child_spec(_) do
    %{
      id: __MODULE__,
      start: {__MODULE__, :start_link, []},
      type: :supervisor
    }
  end
```

dynamic_workers/lib/todo/database.ex (2 of 2)

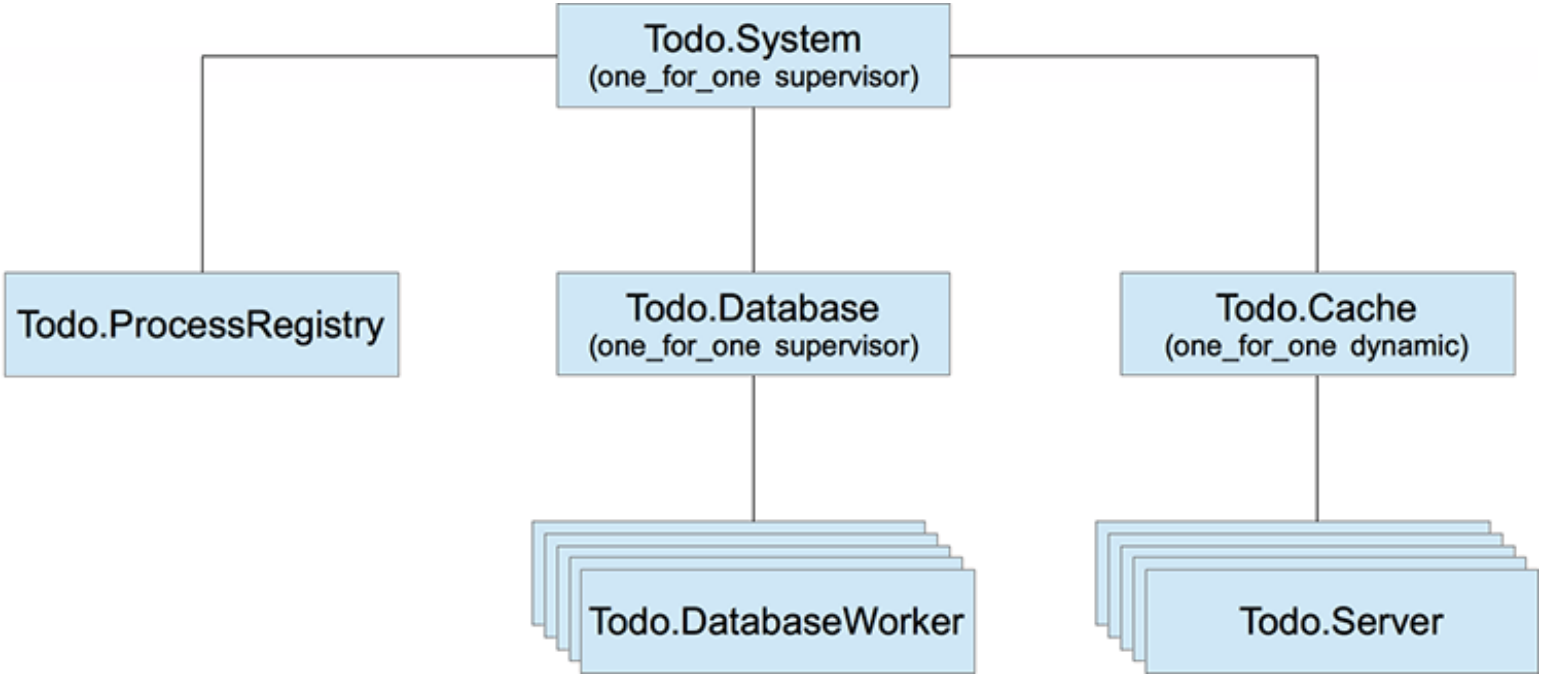
```
 def store(key, data) do
    key
    |> choose_worker()
    |> Todo.DatabaseWorker.store(key, data)
  end

  def get(key) do
    key
    |> choose_worker()
    |> Todo.DatabaseWorker.get(key)
  end

  defp choose_worker(key) do
    :erlang.phash2(key, @pool_size) + 1
  end
end
```

dynamic_workers/lib/todo/cache.ex

```
defmodule Todo.Cache do
  def start_link() do
    IO.puts("Starting to-do cache")
    DynamicSupervisor.start_link(name: __MODULE__, strategy: :one_for_one)
  end

  def child_spec(_arg) do
    %{
      id: __MODULE__,
      start: {__MODULE__, :start_link, []},
      type: :supervisor
    }
  end

  # The function we called from the prompt line in demo to create a todo list
  def server_process(todo_list_name) do
    case start_child(todo_list_name) do
      {:ok, pid} -> pid
      {:error, {:already_started, pid}} -> pid
    end
  end

  defp start_child(todo_list_name) do
    DynamicSupervisor.start_child(__MODULE__, {Todo.Server, todo_list_name})
  end
end
```

Break

Where we are?



- We have studied the top 4 single processes in the supervisory tree.
- Next, we will study the child processes in the bottom of the tree.

dynamic_workers/lib/todo/server.ex (1 of 2)

```
defmodule Todo.Server do
  # See https://zohaib.me/use-in-elixir-explained/
  use GenServer, restart: :temporary

  def start_link(name) do
    GenServer.start_link(Todo.Server, name, name: via_tuple(name))
  end

  def add_entry(todo_server, new_entry) do
    GenServer.cast(todo_server, {:add_entry, new_entry})
  end

  def entries(todo_server, date) do
    GenServer.call(todo_server, {:entries, date})
  end

  defp via_tuple(name) do
    Todo.ProcessRegistry.via_tuple({__MODULE__, name})
  end
```

dynamic_workers/lib/todo/server.ex (2 of 2)

```
  @impl GenServer
  def init(name) do
    IO.puts("Starting to-do server for #{name}")
    {:ok, {name, Todo.Database.get(name) || Todo.List.new()}}
  end

  @impl GenServer
  def handle_cast({:add_entry, new_entry}, {name, todo_list}) do
    new_list = Todo.List.add_entry(todo_list, new_entry)
    Todo.Database.store(name, new_list)
    {:noreply, {name, new_list}}
  end

  @impl GenServer
  def handle_call({:entries, date}, _, {name, todo_list}) do
    {
      :reply,
      Todo.List.entries(todo_list, date),
      {name, todo_list}
    }
  end
end
```

dynamic_workers/lib/todo/database_worker.ex (1 of 2)

```
defmodule Todo.DatabaseWorker do
  use GenServer

  def start_link({db_folder, worker_id}) do
    IO.puts("Starting database worker #{worker_id}")

    GenServer.start_link(
      __MODULE__,
      db_folder,
      name: via_tuple(worker_id)
    )
  end

  def store(worker_id, key, data) do
    GenServer.cast(via_tuple(worker_id), {:store, key, data})
  end

  def get(worker_id, key) do
    GenServer.call(via_tuple(worker_id), {:get, key})
  end

  defp via_tuple(worker_id) do
    Todo.ProcessRegistry.via_tuple({__MODULE__, worker_id})
  end
```

dynamic_workers/lib/todo/database_worker.ex (2 of 2)

```
  @impl GenServer
  def init(db_folder) do
    {:ok, db_folder}
  end

  @impl GenServer
  def handle_cast({:store, key, data}, db_folder) do
    db_folder
    |> file_name(key)
    |> File.write!(:erlang.term_to_binary(data))

    {:noreply, db_folder}
  end

  @impl GenServer
  def handle_call({:get, key}, _, db_folder) do
    data =
      case File.read(file_name(db_folder, key)) do
        {:ok, contents} -> :erlang.binary_to_term(contents)
        _ -> nil
      end

    {:reply, data, db_folder}
  end

  defp file_name(db_folder, key) do
    Path.join(db_folder, to_string(key))
  end
end
```

dynamic_workers/lib/todo/list.ex (1 of 2)

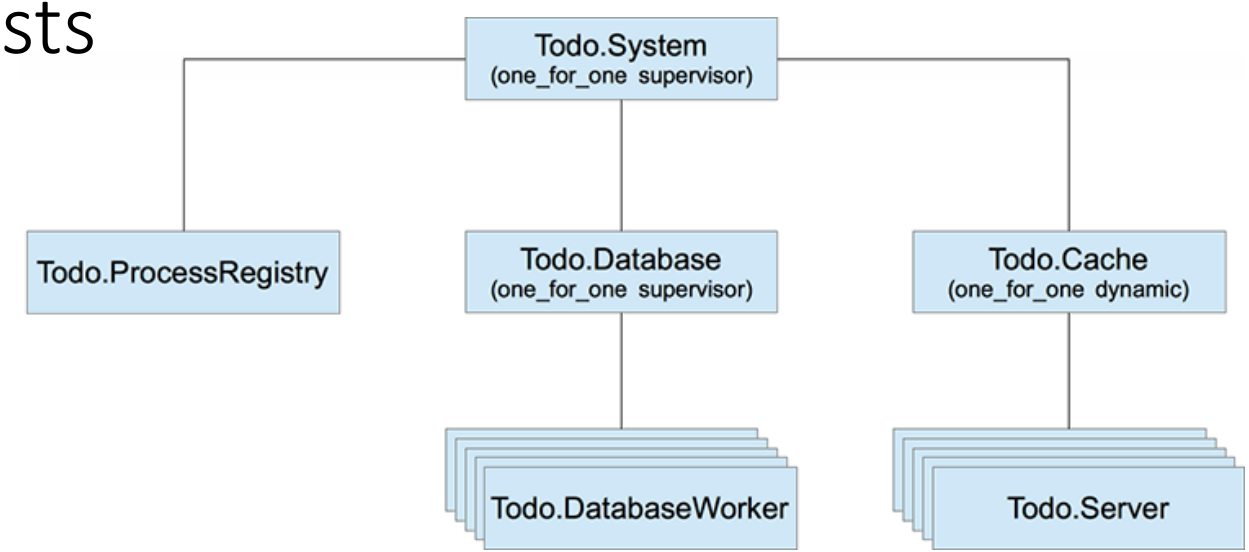
```
defmodule Todo.List do
  defstruct auto_id: 1, entries: %{}

  def new(entries \\ []) do
    Enum.reduce(
      entries,
      %Todo.List{},
      &add_entry(&2, &1)
    )
  end

  def size(todo_list) do
    Map.size(todo_list.entries)
  end

  def add_entry(todo_list, entry) do
    entry = Map.put(entry, :id, todo_list.auto_id)
    new_entries = Map.put(todo_list.entries, todo_list.auto_id, entry)

    %Todo.List{todo_list | entries: new_entries, auto_id: todo_list.auto_id + 1}
  end

  def entries(todo_list, date) do
    todo_list.entries
    |> Stream.filter(fn {_, entry} -> entry.date == date end)
    |> Enum.map(fn {_, entry} -> entry end)
  end
```

dynamic_workers/lib/todo/list.ex (2 of 2)

```
  def update_entry(todo_list, %{} = new_entry) do
    update_entry(todo_list, new_entry.id, fn _ -> new_entry end)
  end

  def update_entry(todo_list, entry_id, updater_fun) do
    case Map.fetch(todo_list.entries, entry_id) do
      :error ->
        todo_list

      {:ok, old_entry} ->
        new_entry = updater_fun.(old_entry)
        new_entries = Map.put(todo_list.entries, new_entry.id, new_entry)
        %Todo.List{todo_list | entries: new_entries}
    end
  end

  def delete_entry(todo_list, entry_id) do
    %Todo.List{todo_list | entries: Map.delete(todo_list.entries, entry_id)}
  end
end
```

Break

Automated Tests



- We have two test suites.
- todo_cache_test.exs – testing how the system works for the multiple todo lists
- todo_list_test.exs – testing how a todo list operates

dynamic_workers/test/todo_cache_test (1 of 2)

```
defmodule TodoCacheTest do
  use ExUnit.Case

  setup_all do
    {:ok, todo_system_pid} = Todo.System.start_link()
    {:ok, todo_system_pid: todo_system_pid}
  end

  test "server_process" do
    bob_pid = Todo.Cache.server_process("bob")

    assert bob_pid != Todo.Cache.server_process("alice")
    assert bob_pid == Todo.Cache.server_process("bob")
  end

  test "to-do operations" do
    jane = Todo.Cache.server_process("jane")
    Todo.Server.add_entry(jane, %{date: ~D[2018-12-19], title: "Dentist"})
    entries = Todo.Server.entries(jane, ~D[2018-12-19])

    assert [%{date: ~D[2018-12-19], title: "Dentist"}] = entries
  end
```

dynamic_workers/test/todo_cache_test (2 of 2)

```
 test "persistence", context do
    john = Todo.Cache.server_process("john")
    Todo.Server.add_entry(john, %{date: ~D[2018-12-20], title: "Shopping"})
    assert 1 == length(Todo.Server.entries(john, ~D[2018-12-20]))

    Process.exit(john, :kill)

    entries =
      "john"
      |> Todo.Cache.server_process()
      |> Todo.Server.entries(~D[2018-12-20])

    assert [%{date: ~D[2018-12-20], title: "Shopping"}] = entries
  end
end
```

dynamic_workers/test/todo_list_test (1 of 3)

```
defmodule TodoListTest do
  use ExUnit.Case, async: true

  test "empty list" do
    assert Todo.List.size(Todo.List.new()) == 0
  end

  test "entries" do
    todo_list =
      Todo.List.new([
        %{date: ~D[2018-12-19], title: "Dentist"},
        %{date: ~D[2018-12-20], title: "Shopping"},
        %{date: ~D[2018-12-19], title: "Movies"}
      ])

    assert Todo.List.size(todo_list) == 3
    assert todo_list |> Todo.List.entries(~D[2018-12-19]) |> length() == 2
    assert todo_list |> Todo.List.entries(~D[2018-12-20]) |> length() == 1
    assert todo_list |> Todo.List.entries(~D[2018-12-21]) |> length() == 0

    titles = todo_list |> Todo.List.entries(~D[2018-12-19]) |> Enum.map(& &1.title)
    assert ["Dentist", "Movies"] = titles
  end
```

dynamic_workers/test/todo_list_test (2 of 3)

```
test "add_entry" do
   todo_list =
     Todo.List.new()
     |> Todo.List.add_entry(%{date: ~D[2018-12-19], title: "Dentist"})
     |> Todo.List.add_entry(%{date: ~D[2018-12-20], title: "Shopping"})
     |> Todo.List.add_entry(%{date: ~D[2018-12-19], title: "Movies"})

   assert Todo.List.size(todo_list) == 3
   assert todo_list |> Todo.List.entries(~D[2018-12-19]) |> length() == 2
   assert todo_list |> Todo.List.entries(~D[2018-12-20]) |> length() == 1
   assert todo_list |> Todo.List.entries(~D[2018-12-21]) |> length() == 0

   titles = todo_list |> Todo.List.entries(~D[2018-12-19]) |> Enum.map(& &1.title)
   assert ["Dentist", "Movies"] = titles
 end
```

dynamic_workers/test/todo_list_test (3 of 3)

```
  test "update_entry" do
 todo_list =
      Todo.List.new()
      |> Todo.List.add_entry(%{date: ~D[2018-12-19], title: "Dentist"})
      |> Todo.List.add_entry(%{date: ~D[2018-12-20], title: "Shopping"})
      |> Todo.List.add_entry(%{date: ~D[2018-12-19], title: "Movies"})
      |> Todo.List.update_entry(2, &Map.put(&1, :title, "Updated shopping"))

    assert Todo.List.size(todo_list) == 3
    assert [%{title: "Updated shopping", date: ~D[2018-12-20], id: 2}] ==
                           Todo.List.entries(todo_list, ~D[2018-12-20])
  end

  test "delete_entry" do
    todo_list =
      Todo.List.new()
      |> Todo.List.add_entry(%{date: ~D[2018-12-19], title: "Dentist"})
      |> Todo.List.add_entry(%{date: ~D[2018-12-20], title: "Shopping"})
      |> Todo.List.add_entry(%{date: ~D[2018-12-19], title: "Movies"})
      |> Todo.List.delete_entry(2)

    assert Todo.List.size(todo_list) == 2
    assert Todo.List.entries(todo_list, ~D[2018-12-20]) == []
  end
end
```

Break

Marking Criteria of Final Exam Project

- You are required to modify the sample project in the 4 areas.
- Originality makes up 10% of your score.

Area	Weight (%)
Functionality	35
Efficiency or capacity	15
Fault tolerance	15
Automated testing	25
Originality	10

Functionality Suggestions

1. Create a nice report summary (daily or weekly)
2. Add time in addition to date
3. Track time spent as well as planning
4. User authentication
5. Sharing todo list (read and/or write)

A. Choose your modifications wisely; some may take more effort than others.
B. You can do more than one change if you have time.

Efficiency or capacity

- Efficiency means doing it faster
- Capacity means the ability to handle more workload or data
- Sometimes they can be achieved by having more processes working in parallel
- Sometimes they can be achieved by revising the algorithms used

Fault Tolerance

- The sample project already possesses some fault tolerance as demonstrated in the killing of a todo server.
- Can you test the fault tolerance to other kind of processes?
- If it is not fault tolerant, can you improve it?
- Is there limitation in the existing fault tolerance? For example, a limit of 3 failures in 5 minutes. Can you find the limitation and relax it a little.

Automated Testing

- This is the area that you can enhance most easily because the sample test code is not comprehensive.
- For example, it tested the fault tolerance of a crashed todo server in a minimal way. It did not test the fault tolerance of a crashed database worker.

Originality

- If your modification is different from classmates’ work, you are considered original.
- Will good idea be similar any way? Even if functionality is the same, you design the API more elegant than others.

Submission Requirements

1. All changes
   A. must have an objective and explanation of how it works especially if the code is difficult to understand.
   B. must be demonstrated through either screens of running at the prompt line or automated tests
   C. must be identified by the line numbers being replaced with file name
2. Must my modifications be an improvement?
   A. Preferably your modification is a clear improvement.
   B. If you cannot find an improvement in an area, at least make some changes and explain the effect of your changes.