What happens to an Android thread after the Activity that created it is destroyed?

Asked 10 years, 2 months ago Active 9 years, 10 months ago Viewed 7k times

19

4

In my Android app, the main Activity will sometimes start a Thread to load data from a server. This thread modifies the app's database and edits some important files. AFAIK, it would appear that this thread continues to execute. What will happen to this thread if the Android gets into a low memory situation and decides to kill the entire Application? Will there ever be a situation were this thread might die prematurely? If so, is there any way I can see that the thread is being killed, and do something about it?

I'm asking because this thread modifies important data in the database, and if it is suddenly killed, the application could stop functioning properly.

java android multithreading android-activity

Share Improve this question Follow

asked Jun 1 '11 at 22:22
Zarjio
1,019 ● 2 ● 11 ● 21

4 Answers

Active Oldest Votes

16

> AFAIK, it would appear that this thread continues to execute.

This is true but you have no guarantee of how long the thread will stay alive.

> What will happen to this thread if the Android gets into a low memory situation and decides to kill the entire Application?

This is actually a fairly rare case in my experience but it will depend on the device's available memory and the user's behaviour, for example they use the device heavily and start multiple apps.

> Will there ever be a situation were this thread might die prematurely?

Yes

> If so, is there any way I can see that the thread is being killed, and do something about it?

No

> I'm asking because this thread modifies important data in the database, and if it is suddenly killed, the application could stop functioning properly.

What you describe could be classed as something which is 'mission critical'. As the other two answers have pointed out, a Service would be a more robust way of doing things as a Service is one of the last things to be 'killed' in a low memory situation. Using START_REDELIVER_INTENT may help in resuming what it was doing.

In any case, if you have a 'mission critical' operation, you need to design your code for full recovery such as the use of transactions and the possibility of rollbacks in case of errors.

Share Improve this answer Follow

answered Jun 1 '11 at 23:41
Squonk
47.8k ● 18 ● 100 ● 134

Sure, there is something you can do about it. But not once you've allowed app to be destroyed - act sooner than `onDestroy` - In `onStop` set a flag that thread can poll, to know that there is a risk of app going away. Clear that flag in `onResume`. Thread does its work in as short chunks as is possible, and if it sees that flag, it ends itself. At end of each chunk, it updates status of its progress in file, so can restart correctly. Also in `onStop`, don't return until `!thread.isAlive` or 1 second - loop with test and counter and sleep 100 ms at a time, no more than 10 times. – ToolmakerSteve Sep 22 '16 at 12:52

Thread also needs to check flag after any long delay (e.g. response from server), and before doing critical actions (updating local files/database). It never begins a critical section with that flag set. As long as critical sections are no more than 100-200 ms, I think you are safe in practice. (Not claiming this is good approach for mission critical app, but good enough for most of us.) Notice that the essence of my approach is to do work *while your app is still alive* - by delaying `onStop` for up to 1 second. For many situations, this is all that is needed. – ToolmakerSteve Sep 22 '16 at 12:57

UPDATE: My suggestion of delaying up to 1 second is way too long to be appropriate for `onStop`. Need to research how long is appropriate - what is Android's definition of a reasonable amount of time. Note that if there is any UI call where Android *should* be somewhat lenient, it is `onStop`, since that is an app's last chance to be sure they have preserved whatever they care about. – ToolmakerSteve Sep 22 '16 at 13:04

5

It sounds like you should move the db updating to a Service. Once an Activity goes to the background, Android assumes its process can be killed off if necessary and restarted later without ill effect. See [Application Fundamentals](#) for more info.

Share Improve this answer Follow

answered Jun 1 '11 at 22:26
Ted Hopp
224k ● 48 ● 373 ● 491

5

You should be using a service:

> Because a process running a service is ranked higher than a process with background activities, an activity that initiates a long-running operation might do well to start a service for that operation, rather than simply create a worker thread—particularly if the operation will likely outlast the activity.

Take a look here: http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html#Lifecycle (where the text I pasted originates)

Share Improve this answer Follow

answered Jun 1 '11 at 22:32
dimi
1,436 ● 2 ● 15 ● 26

5

I might be digging out an old thread, but nobody mentioned one important thing.

Every time you use a database and modify multiple rows, you should make use of transactions to ensure that data stays valid in case of any kind of failure (which might be for example a termination of a thread, a socket exception etc.).

```
    try{
        db.beginTransaction();

        //Do whatever you need to do...

        db.setTransactionSuccessful();
    }catch(SQLiteException e){
        Log.e("SQLite","Error while updating rows: " + e.getMessage());
    }finally{
        db.endTransaction(); //Commit (if everything ok) or rollback (if any 
error occured).
        db.close(); //Close databse;
    }
```

Share Improve this answer Follow

answered Oct 12 '11 at 19:23
bart.g
51 ● 1 ● 1