How can we prevent a Service from being killed by OS?

I am using `Service` in my application and it needs to run until my application is uninstalled, but the problem is it gets killed by OS.

How can we prevent it from being killed by OS? Or if it gets killed can we restart that service again through programmatically?

android android-service

edited Apr 29 '15 at 4:39
Andrew T.
4,041 5 25 47

asked Mar 14 '12 at 6:15
Rahul
357 1 6 13

11 Answers

You may run the service in the foreground using startForeground().

> A foreground service is a service that's considered to be something the user is actively aware of and thus not a candidate for the system to kill when low on memory.

Note: This still does not absolutely guarantee that the service won't be killed under extremely low memory conditions. It only makes it less likely to be killed.

edited May 23 '17 at 12:10

Community ♦
1 1

answered Mar 14 '12 at 6:28

Dheeraj V.S.
12k 6 45 82

3 Thanks for ur reply Dheeraj but i dont need notification in service is there any other way to run service without gets killed by os.. Did you know Wake_Lock.. – Rahul Mar 14 '12 at 6:58

2 i have merged code in my service what u have given me a link startForeground(). but my service is also gets killed by os after sometimes . – Rahul Mar 14 '12 at 8:56

1 it is not working Dheeraj... – Rahul Mar 14 '12 at 10:47

@Rahul u got any solution for this? – PankajAndroid Nov 13 '13 at 5:54

5 It's just the way Android handles low memory situations. If you really needed Android not to kill your service, your best bet would be to make it a System App, or make your service return `START_STICKY` in your `onStartCommand()` method. This way, if your service gets killed, then it will be queued to restart automatically. – jaytj95 Jul 18 '14 at 17:09

I've been puzzled by the same issue to yours recently.but now,I've found a good solution. First of all,you should know that, even your service was killed by OS, the onCreate method of your service would be invoked by OS in a short while.So you can do someting with the onCreate method like this:

```
@Override
public void onCreate() {
    Log.d(LOGTAG, "NotificationService.onCreate()...");
    //start this service from another class
    ServiceManager.startService();
}
@Override
public void onStart(Intent intent, int startId) {
    Log.d(LOGTAG, "onStart()...");
    //some code of your service starting,such as establish a connection,create a
TimerTask or something else
}
```

```
public static void startService() {
    Log.i(LOGTAG, "ServiceManager.startSerivce()...");
    Intent intent = new Intent(NotificationService.class.getName());
    context.startService(intent);
}
```

However, this solution is just available for the situation of your service being killed by GC.Sometimes our service might be killed by user with Programme Manager.In this situation,your prosses will be killed,and your service will never been re-instantiated.So your service can not be restarted. But the good news is,when the PM kill your service,it will call your onDestroy method.So we can do something with that method.

```
    @Override
public void onDestroy() {
    Intent in = new Intent();
    in.setAction("YouWillNeverKillMe");
    sendBroadcast(in);
    Log.d(LOGTAG, "onDestroy()...");
}
```

The string of "YouWillNeverKillMe" is a custom action. The most important thing of this method is,don't add any code before send the broadcast.As system will not wait for completion of onDestroy(),you must send out the broadcast as soon as posible. Then regist a receiver in manifast.xml:

```
<receiver android:name=".app.ServiceDestroyReceiver" >
        <intent-filter>
            <action android:name="YouWillNeverKillMe" >
            </action>
        </intent-filter>
    </receiver>
```

Finally,create a BroadcastReceiver,and start your service in the onReceive method:

```
@Override
public void onReceive(Context context, Intent intent) {
    Log.d(LOGTAG, "ServeiceDestroy onReceive...");
    Log.d(LOGTAG, "action:" + intent.getAction());
    Log.d(LOGTAG, "ServeiceDestroy auto start service...");
    ServiceManager.startService();
}
```

answered Jun 20 '12 at 8:54

Alaowan
233 2 3

Tested on Android 4.0.4 and looks promising so far! – Alexandre Lavoie Apr 12 '14 at 6:51

3 @Alaowan: You have placed the restart code in different places.If the service is killed by GC or for that matter PM,then in either case if onDestroy is called,then isnt it much better to called startService in the onDestroy itself,as mentioned in the answer by [at]androidiseverythingforme ? – Basher51 Oct 7 '14 at 6:49

2 Your code to prevent users from stopping the service does more harm than good since I think users should have control over their own devices; it's annoying for them if you try to stop them from manually killing your services. I know there could be some situations where this is appropriate, but this question is about stopping the *OS* from killing the service. – Sam May 8 '15 at 8:43

2 I can't believe this answer got 13 votes. – Daniele Ricci Nov 4 '15 at 19:58

1 Please do not upvote this answer. This is not correct. – blueether Jul 5 '16 at 2:57

Override method `onStartCommand()` in your service class and simply return `START_STICKY` (as suggested by "Its not blank"). That's all you need. If the process that runs your service gets killed (by a low memory condition for example), the Android system will restart it automatically (usually with some delay, like 5 seconds).

Don't use `onStart()` anymore as suggested in another answer, it's deprecated.

edited Aug 11 '13 at 22:28

IInspectable
21.8k 3 34 75

answered Aug 11 '13 at 22:02

Chris
841 8 6

Note that this is the default behaviour for services anyway, so I don't think you even need to explicitly do this. – Sam May 8 '15 at 8:27

use

```
    //**Your code **
    // We want this service to continue running until it is explicitly
    // stopped, so return sticky.
    return START_STICKY;
}
```

ref Documentation lifecycle of Service.

Edit added method.

edited Mar 22 '16 at 10:15

Ziem
3,634 7 37 69

answered Mar 14 '12 at 6:35

Its not blank
2,320 15 31

i didnt get it Vincent.. – Rahul Mar 14 '12 at 7:00

1 Vincent i have used your method but my service is again gets killed by os after sometimes..is there any other way to do that. – Rahul Mar 14 '12 at 10:43

I use START_NOT_STICKY – yozawiratama Jul 21 '14 at 4:07

2 This seems misleading to me; I don't think `START_STICKY` stops the service from being killed. I think it causes it to restart if it crashes or is killed by the OS. – Sam May 8 '15 at 8:28

I found another solution of the problem which gurantees that your service will be always alive. In my case, this scheme resloves also the problem with FileObserver, which stops work after some period of time.

1. Use an activity (StartServicesActivity) to start the service (FileObserverService) as Foreground service.
2. Use BroadcastReceiver class (in example CommonReceiver) to restart your service in some special situations and in case it was killed.

I used this code in my app "Email Pictures Automatically"
https://play.google.com/store/apps/details?id=com.alexpap.EmailPicturesFree

Here is CommonReceiver class.

```
    public void onReceive(Context paramContext, Intent paramIntent)
    {
        paramContext.startService(new Intent(paramContext,
FileObserverService.class));
    }
}
```

Here is its definition in AndroidManifest.xml just before application closing tag.

```
<receiver android:name="com.alexpap.services.CommonReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.USER_PRESENT"/>
    </intent-filter>
</receiver>
```

Start service in StartServicesActivity activity.

```
Intent iFileObserver = new Intent(StartServicesActivity.this,
FileObserverService.class);
StartServicesActivity.this.startService(iFileObserver);
```

Here is onStartCommand() method of the service.

```
public int onStartCommand(Intent intent, int flags,  int startId) {

    int res = super.onStartCommand(intent, flags, startId);

    /*** Put your code here ***/

    startServiceForeground(intent, flags, startId);

    return Service.START_STICKY;
}

public int startServiceForeground(Intent intent, int flags, int startId) {

    Intent notificationIntent = new Intent(this, StartServicesActivity.class);
    notificationIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);
```

```
    Notification notification = new NotificationCompat.Builder(this)
        .setContentTitle("File Observer Service")
        .setContentIntent(pendingIntent)
        .setOngoing(true)
            .build();

    startForeground(300, notification);

    return START_STICKY;
}
```

I tested this code using Task Killer app, and each time the service was killed, it was restarted again almost immediately (performs onStartCommand()). It is restarted also each time you turn on the phone and after rebooting. I use this code in my application, which emails every picture you take with your phone to predefinde list of emails. The sending email and list of receiving emails are set in another activity and are stored in Shared Preferences. I took about 100 pictures in several hours and all they were sent properly to receiving emails.

edited Jun 30 '15 at 2:55

answered Jun 10 '15 at 19:07

AlexPap

111 1 3

Not workign on Lenovo K3 Note. Problem is that when app is killed from launcher, onTaskRemoved / onDestroy are not called. Also, once app is killed, the broadcast receivers are also not invoked for the event filters indicated above (boot_complete/connectivity_change), etc. The best you can do in these scenarios is to start as foreground service, as mentioned by @Dheeraj – blueether Jul 5 '16 at 3:17

```
@Override
public void onDestroy() {

    super.onDestroy();
    startService(new Intent(this, YourService.class));

}
```

write above code in your service and your service will never stop even user want to destroy it or they want to kill it it will never kill untill your app not get uninstall from your device

Android is everything for me

2,883 3 18 30

yup i used this line and it works fine for me this is d exact answer unfortunately i havent more point to vote up – Android is everything for me May 13 '14 at 7:26

1 @Basher51, no; from my experience, `onDestroy` isn't always called when a service is killed. – Sam Nov 14 '14 at 23:32

3 @Basher51, 1. When the app is upgraded; 2. When the OS kills it due to low RAM; 3. When the app crashes. – Sam May 8 '15 at 8:22

1 In my opinion, this code makes things worse. When the OS kills your service, this doesn't work. When the user kills your service, this *does* work, which I think is a bad thing since it takes control away from the user. Also, when you try to stop the service in your code using `stopService`, this prevents it from working, which I think is also a bad thing. – Sam May 8 '15 at 8:28

1 This works well, just what I was looking for and much simpler than other answers, thanks. Even if I kill it, it again starts :) – Shahbaz Talpur Oct 23 '16 at 17:20

You can try to start your service repeatedly, for example every 5 sec. This way, when your service is running, it will perform onStartCommand() every 5 sec. I tested this scheme and it is very reliable, but unfortunately it increases slightly phone overhead. Here is the code in your activity where you start the service.

```
Intent iFileObserver = new Intent(StartServicesActivity.this,
FileObserverService.class);
PendingIntent pendingIntentFileObserver =
PendingIntent.getService(StartServicesActivity.this, 0, iFileObserver, 0);
AlarmManager alarmManager = (AlarmManager)getSystemService(ALARM_SERVICE);
Date now = new Date();

    //start every 5 seconds
alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, now.getTime(), 5*1000,
pendingIntentFileObserver);
```

And here is onStartCommand() of the service.

```
//class variable
public static boolean isStarted = false;
```

```
    int res = super.onStartCommand(intent, flags, startId);

    //check if your service is already started
    if (isStarted){      //yes - do nothing
        return Service.START_STICKY;
    } else {             //no
        isStarted = true;
    }


    /**** the rest of your code  ***/

    return Service.START_STICKY;

}
```

edited Jun 11 '15 at 4:32

answered Jun 9 '15 at 7:04

AlexPap
111 1 3

As far i know, `onDestroy()` will be called only when the service is explicitly stopped(Force Stop). But this method won't get called in case the service gets killed by OS/swiping the Recent Apps list. In those cases another event handler named `onTaskRemoved(Intent)` gets called. This is due to a defect in Android 4.3-4.4 as per the link here. Try using the below code:-

```
public void onTaskRemoved(Intent intent){
super.onTaskRemoved(intent);
Intent intent=new Intent(this,this.getClass());
startService(intent);
}
```

edited Jul 20 '15 at 11:33

answered Jul 20 '15 at 9:51

Karthik
116 1 1 9

Thank You For Help !! My App Is Running..... – Anirban Bhui Apr 27 '16 at 6:32

First create service in another process, and write broadcaster which runs in recursion in time intervals

```
protected CountDownTimer rebootService = new CountDownTimer(9000, 9000) {


    @Override
    public void onTick(long millisUntilFinished) {


    }

    @Override
    public void onFinish() {

        sendBroadcast(reboot);
        this.start();
        Log.d(TAG, "rebootService sending PREVENT AUTOREBOT broadcast");


    }

};
```

After that register broadcast receiver in main process also with timer recursion that is launched after first broadcast from service arrived

```
protected static class ServiceAutoRebooter extends BroadcastReceiver {

    private static   ServiceAutoRebooter instance = null;
    private  RebootTimer rebootTimer = null;

    private static ServiceAutoRebooter getInstance() {

        if (instance == null) {

            instance = new ServiceAutoRebooter();

        }
        return instance;

    }
```

```
    private Context _context;
    private Intent _service;


    public RebootTimer(long millisInFuture, long countDownInterval) {
        super(millisInFuture, countDownInterval);
    }


    @Override
    public void onTick(long millisUntilFinished) {

    }


    @Override
    public void onFinish() {

        _context.startService(_service);
        this.cancel();
        Log.d(TAG, "Service AutoRebooted");


    }


}


@Override
public void onReceive(Context context, Intent intent) {

    if (rebootTimer == null) {
        Log.d(TAG, "rebootTimer == null");

        rebootTimer = new RebootTimer(10000, 10000);
        rebootTimer._context = context;

        Intent service = new Intent(context, SomeService.class);
        rebootTimer._service = service;
        rebootTimer.start();

    } else {

        rebootTimer.cancel();
        rebootTimer.start();
```

```
        }

    }

}
```

Service will be auto-rebooted if time at RebootTimer (main process) expires, which means that "PREVENT AUTOREBOT" broadcast from service hasn't arrived

answered Mar 2 '16 at 12:17

Emir Hadzimahmutovic

1 3

---

i found a solution .... late answer but i wanted to answer...

we can send a broadcast in the ondestroy of the service and create a receiver that receives the broadcast and starts the service again.... when it is destroyed by any reasons...

answered Jan 15 '13 at 18:39

user1106888

79 6

This isn't true; it only seems to cover the cases when the user or developer stop the service in my experience. – Sam May 8 '15 at 8:30

---

pls try following:

```
final Messenger mMessenger = new Messenger(new IncomingHandler());
class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            default:
                super.handleMessage(msg);
            }
```

```
@Override
public void onCreate() {
    super.onCreate();
    makeServiceForeground();
}


@Override
public IBinder onBind(Intent arg0) {
    return mMessenger.getBinder();
}


private void makeServiceForeground() {
    IActivityManager am = ActivityManagerNative.getDefault();
    try {
        am.setProcessForeground(onBind(null), android.os.Process.myPid(), true);
    } catch (RemoteException e) {
        Log.e("", "cant set to foreground" + e.toString());
    }
}
```

also need add in manifest.xml

```
<uses-permission android:name="android.permission.SET_PROCESS_LIMIT"/>
```

answered Apr 17 '13 at 17:53

m0narX
1 3

Have you tested this code yourself? – Iwo Banas Jun 21 '13 at 7:50

yeah)) but on system apps – m0narX Aug 24 '13 at 13:11

2 This is overcomplicated. You can just use the public method, `Service.startForeground` rather than these private APIs. And foreground processes can still be killed by the OS in low memory situations. – Sam May 8 '15 at 8:32